SOGETI

Workbook TMap® Suite

Studying for the TMap® Suite Certification

Sogeti Nederland B.V.
Vianen, 2016

Contents

Introduction

Why this workbook?

Since its introduction in 1995 TMap (Test Management Approach) has grown to become the standard for the structured testing of software. This position was further reinforced with the arrival of TMap NEXT® in 2006. Test Managers and Testers validate their professionalism by obtaining the EXIN certification TMap NEXT® Test Master or Test Engineer.

The continuous improvement of the method has led to the development of the TMap Suite in 2014. The TMap Suite consists of the following components:

- The new approach: **TMap HD - Human Driven**. A quality-driven test method for modern agile organizations. This is described in the novel "Neil's Quest for Quality".
- The new **TMap.net website**. This new website contains the Building Blocks of TMap. They can be used to build your own test method.
- **TMap NEXT®** is a method of testing for organizations using traditional development methods such as waterfall.

This workbook has been developed as support for obtaining the EXIN certification TMap® Suite Test Manager and Test Engineer. It is a compilation from several sources of literature (from TMap HD, the website and TMap NEXT), which together constitute the material for the exams. This is not a new TMap book and it contains no new information compared to previous books. The order of the sources in this book does not imply the importance of the various subjects, nor does it indicate in which order the subjects should be studied. It is only meant as a tool for examinees and to clarify which parts of the TMap Suite are included in the material for the exam. In order to understand the connection between the various subjects, it is imperative to refer to the original books (and website) or attend a training that leads up to this certification.

In the first chapter TMap HD is discussed. This chapter contains the relevant parts of this novel. The second chapter is about the Building Blocks of TMap. In the third chapter we take a look at the website TMap.net. And finally, the relevant information from the TMap Next book is included in the fourth and final chapter.

Chapter 1 Mr. Mikkel's Musings

The reflections below are taken from the book "Neil's Quest for Quality".

1.1 Mr. Mikkel's Musings (1) on Building Blocks

"A journey of a thousand miles begins with a single step."
Lao Tze

Typically, when I'm coaching someone on quality and testing, I find that they can be overwhelmed if they are presented with a whole method for quality and test all at once. It is much easier to present each part of the method independently and acquaint them with one part before introducing the next. Often it is best to start with the parts that are most important to their situation, have them learn and implement these, and then start on the next one.

The same thing works on a larger scale as well. When organizations want to change to a better quality and testing method, it is much easier for them to learn one part well, implement that part to solve a particular problem, and then look for a next part. When an organization is confronted with a whole new process all at once, this often leads to poor understanding of this process. In those cases, many steps and tools are not very well understood. This leads to situations where following the method becomes the goal rather than solving the problem at hand. This leads to the 'in-name-only variants' of standard methods.

Furthermore, every organization is different and has different needs for its testing method. Are you Agile, or more traditional? Do you have to meet very formal quality standards or not? Do you have very experienced people in your organization or are you a young and eager company that has to learn a lot?

All those things and more have an influence on how you model the testing and quality method for your organization. This means that every organization has its own optimum method. A method that can be optimal for an organization at one point in time, can become less optimal when something changes in the situation. For instance, the introduction of a new tool that makes it easier to test certain things may demand a change in the method.

What people and organizations find very helpful is to build up the method gradually themselves, with the aid of Building Blocks. A Building Block is a process step or a tool or a role that can solve a particular testing and quality problem in your organization. A Building Block can also be fitted into the existing method, or moved around within the method. For instance, a specific test may be shifted to a point earlier in the lifecycle to detect certain faults earlier in the process.

You can also make your own Building Blocks. If your organization has to conform to specific standards, for example, you can create a special building block to check whether or not these standards are being met.

A great starting collection of Building Blocks of TMap HD can be found on www.tmap.net. Feel free to use them as you please and adapt them to your specific situation!

I can imagine that you would like more inspiration on the topic of how to link Building Blocks together. This is all part of the TMap Suite and can be found on tmap.net as well.

1.2 Mr. Mikkel's Musings (2) on the elements

"It's elementary, my dear Watson"
Sherlock Holmes, in the film 'The Return of Sherlock Holmes'

When I coached Neil, I introduced to him the 5 elements of quality-driven testing.

These elements have two purposes. On the one hand, they are elements of evolution of the quality and testing profession. The profession of quality and testing is changing and these elements indicate the direction of this change.

The elements also helped Neil to make choices, to achieve better results and find answers to questions such as:

- What is the best test strategy?
- How can I test more and faster?
- How can I accomplish better quality?
- Which Building Blocks can I use?
- How can I apply the Building Blocks?

Simplify

"Everything must be made as simple as possible. But not simpler."
Albert Einstein

Ever since the start of IT in business, the IT landscape has been growing more and more complex. The implication for testing and quality is that growing complexity requires more testing to address all relations and effects concerning the chain of IT solutions. To end this upward spiral, it is important to simplify, standardize and decouple. Testing and quality as a profession can simplify their activities in step with the simplification of the IT environment.

Apart from simplifying the testing in step with the simplification of the IT landscape, the efficiency of test activities can be improved by keeping the activities small-scale and clear: only those test activities necessary to achieve business value are carried out, but no more than these. Test strategy, test techniques should be chosen in a way that suits the particular situation the best.

Integrate

" Every kind of peaceful cooperation among men is primarily based on mutual trust and only secondarily on institutions such as courts of justice and police."
Albert Einstein

A part of the evolution in IT is the need to integrate. IT complexity is reduced by structuring, simplifying and standardizing IT solutions within a coherent IT landscape, and by integrating IT solutions with business processes.

The process of creating such solutions is under enormous pressure. Integration is one of the answers, where all disciplines involved in the process of creating IT solutions need to cooperate better in order to increase efficiency, speed and quality.

Integration with respect to testing denotes to a shared way of working, with a shared responsibility for quality. Testing is not a stand-alone process and should integrate seamlessly in its environment.

The integration of testing and quality approaches is not new. Testing is a measure to cover a risk, alongside other measures. Sometimes a risk can be covered by extra tests, sometimes it is better to cover it by other quality measures such as pair-programming or test-driven techniques. The point is: in an integrated approach, a risk does not HAVE to be covered by testing.

Industrialize

"The monotony of a quiet life stimulates the creative mind."
Albert Einstein

The standardization of tests provides opportunities for the automation of test execution. Models can be used to automatically generate test cases. In fact, every kind of test activity can by supported by a tool. For example, the planning of a test can be supported by test-management tooling, the specification of a test by model-based methods, there are test-execution tools, and a test environment can be managed with service virtualization and test-data-management tooling. There are also integrated tools, quality suites or lifecycle suites.

The element of Industrialize is very important in improving testing and optimizing quality.

Test tools can be used to test more, more often, and faster. More information on the element of Industrialize can be found in Building Blocks about test tools.

The element of Industrialize implies aspects such as:

- Test automation
- Accelerators
- Standardization
- Re-use
- Test design techniques
- Templates
- Test environments

People

"The important thing is not to stop questioning."
Albert Einstein

Having a method is one thing, applying it is another. Different project management approaches, different company cultures, different quality demands, different environments, etc. They all call for wise use of any method. Without the right people to execute the method, any method will fail.

People need to have the right skills and the right knowledge to perform their jobs. In a quality-driven approach, the appropriate mindset for right-first-time is essential as well. Everybody has a certain notion of quality. Quality professionals are skilled and trained to make the relevant quality aspects tangible and measurable.

Tests can be performed by anyone in an organization, as long as they are helped in this by professional testers who have the critical mindset to test adequately and effectively.

It is the People element that makes it possible to move from testing according to TMap to testing with TMap. For this, you need to have people with a wide knowledge of quality and testing, and the right mindset to be able to apply the Building Blocks in a way that suits their organization. Hence the name TMap HD – Human Driven.

Confidence

"Not everything that can be counted counts, and not everything that counts can be counted."
Albert Einstein

A fifth element emerges from the four elements: Confidence. That is an extra element over and above the others. The 4 elements improve the approach to testing. In conjunction they form the vital basis from which this 5th element arises: Confidence is where they all lead to. It is the 5th element that makes the need for a quality-driven approach unavoidable. The need for reliable IT solutions increases when the dependence on IT increases.

Quality is often defined as 'fit for purpose', but some of my peers state that quality is an irrational sense, and therefore cannot be caught in a definition. The fact is: these 4 elements are indispensible in the creation of confidence in IT solutions.

1.3 Mr. Mikkel's Musings (3) on built-in quality in a quality-driven approach

Perspectives

People usually look at situations from their viewpoint, based on their position, experience and values. The same situation can be described from all these viewpoints, resulting in different stories, sometimes identical, sometimes seemingly contradictory.

Figure 1. Different understandings due to different perspectives

Seabiscuit is a project like many others. Neil described it from his viewpoint, from a quality and testing manager's perspective. This perspective was rather new to him and I was asked to guide him. Owen, Rupert, Francine, Rajiv, Hal and certainly Danielle had different perspectives. It is important to realize that everybody was looking at the same reality.

Quality is built into the process

Rupert's intervention, demanding a quality-driven approach to improve the confidence in IT solutions, made Neil change his viewpoint from the end of the development process to a coordinating position. In a quality-driven approach it is essential that quality be built into the process. Tests are used to monitor the quality during the whole process. Built-in quality is one of the key principles in the Lean approach, as well as continuous improvement, elimination of waste, valuing people. Built-in quality, continuously improved, leads to Right-First-Time, where the outcome of the process fully meets the expectations: fit-for-purpose.

Handling of test and quality issues

Quality refers to the quality of the outcome of the process: the product quality. Process and product quality are strongly linked. Customer value is an essential perspective. That is why a quality-driven approach should also be business-driven. A product is specified and designed for all aspects of the lifecycle. Any deviation in the expected product quality should be detected as soon as possible and should lead to measures. Fixing the fault is not enough, it is essential to improve the process to prevent such defects from happening again. That is how quality is built in. That is how product quality is improved by adjusting the process. An extra test or improving a test (e.g., regression test) can be one of such adjustments, alongside other quality measures. That is also why Neil needed a way to

oversee the whole process and needed a way to influence the total approach even prior to the actual start, supporting the project manager and the teams. Testing is integrated into the development process.

Using the elements to create a quality-driven approach

From his quality and test perspective, Neil used the available Building Blocks in an improved way. I mentioned the elements to do so: Integrate, People, Industrialize and Simplify, where testing leads to Confidence in the developed solutions.

- *Simplify* is always important in all activities, and should be done whenever there is an opportunity. Simplifying the approach by creating short/cycles to keep change small and simple is an example.
- *Integrate* is important to cooperate, have short communication lines, work in a business-driven way. Quality is a shared responsibility.
- *Industrialize* is important to built in many checks during the entire process, in order to detect defects automatically. In Lean manufacturing, these checks are automated as much as possible.
- *People's* attitudes and mindsets contribute greatly to the quality-driven approach. When people are held accountable for delivering quantity or for delivering before a deadline without a defined quality standard, one should not be surprised that the quality is low. When you give people responsibility for a high level of quality and let them decide on their working process, a different outcome can be expected.
- *Confidence* is the main driver of a quality-driven approach.

The development process: Agile - Waterfall

Which development process is actually used is less important. Quality-driven principles can be built into every process. In the first part of the Seabiscuit project, ZBO used a waterfall process; in the second part an Agile process was used. Not because waterfall is bad and Agile is better. Quality-driven elements (simplify, integrate, industrialize, people) can be practiced in all approaches.

Agile has some Lean principles built in, but it is the quality-driven mindset of the people that makes it work. However, Agile is not always a success and not always suitable or applicable, whereas waterfall projects can be very successful.

Long-term effect: project versus staff

It is the mindset focusing on quality by continuous improvement. The ambition is set to zero defects, meaning that there are no deviations from the specified quality criteria. Lean manufacturing has shown that these continuous improvements will result, over time, in rising quality and decreasing costs. A choice is made in favor of the long-term effect. Projects are temporary organizational structures that are not always suitable to obtain long-term effects. That is why a permanent, cross-project organization is important: it consists of a quality staff, who develop and maintain a quality policy, test expertise, a policy on quality and testing tools etc, This policy is passed to projects when they start. In the story, I gave an example of a company that applied this in a project environment independent of the development process used.

Conclusions

So waterfall or Agile is not a choice between good or bad, but an appeal to consider both thoughtfully. It will always be a matter of deliberation. There are even circumstances where a quick and dirty disposable solution will be the best choice. Or the best start, where the disposable is followed by a long-term final solution, as with lower operating costs, for example, or to support a more effective business process. It all depends on the intended purpose.

It is that fit for purpose, also called 'quality', that determines the approach to quality and testing.

Project form, development method, test strategy, etc.: these are all linked and have to be integrated. An adequate integrated method is applicable to every scenario.

An approach for testing is defined by using the Building Blocks in a suitable way.

I am sure that appropriate patterns of Building Blocks, new methods and best practices will arise for all kinds of specific situations. When we all work together and share, we all benefit.

Chapter 2 Building Blocks

These Building Blocks are from the book "Neil's Quest for Quality". These are the parts of the TMAP HD book required for the EXIN exam.

2.1 Building Block 1: Test Manager

What do test managers do? In traditional organizations, they assign people to projects, oversee the testers' progress, provide feedback, and maybe offer some coaching to people who want it. Test managers build trusting relationships with their staff and build up the capacity of the testing group. How does that change with a transition to Agile? Is there still a need for test managers? The answers to these questions are given in the 'Test manager in traditional environments' and 'Test manager in agile environments' Building Blocks. The first will be given directly below this building block, the 'Test manager in Agile environments' can be found in Building Block 12.

In this book we use 'test manager' as a generic term. In practice, you can find many different terms that refer to this role, such as 'test coordinator', 'test leader', 'project leader testing', 'test director', and many more. Sometimes these terms refer to different levels in the organization, when several test coordinators are subordinate to a test manager, for example. We advise you always to make a clear definition of the role and the responsibilities in your specific situation.

2.2 Building Block 2: Test Manager in traditional environments

In traditional organizations, the test manager leads a team of test coordinators and/or testers. Since the test manager oversees the entire testing process, he ought to be able to prevent a fragmented approach. Today's test manager also tries to shift the focus of testing at the end of a project toward other quality measures that can be implemented at the start of a project, such as reviews, inspections, proofs of concept. He or she is the linking pin in drafting the test strategy, bringing all the necessary parties and information together. The test manager is responsible for the planning, management and execution of testing, ensuring that it is on time and on budget and at the right quality, for multiple test varieties. The test manager reports in line with the overall test plan on the progress of the test process and the quality of the test object.

Examples of the test manager tasks:

- Creating the instructions for the test products delivered by the various test varieties
- Checking adherence to the instructions (internal reviews)
- Coordinating the various test activities that apply to the test varieties, such as setting up and managing the technical infrastructure
- Creating guidelines for communication and reporting between the test varieties, and the test process and the suppliers
- Setting up overall test-method-related, technical and functional support
- Keeping the various test plans consistent
- Reporting on the overall test progress, budget and quality of the test object, preferably automated with a test management tool
- Managing expectations of different stakeholders with respect to test progress and quality
- Deploying/hiring (extra) test personnel.

The relationship between the specified roles, the test varieties and the relationships with the other stakeholders in the system development process must be determined and documented. The testing organization is clearly part of the bigger (project) picture. Refer to figure 2 for some examples. In these examples, reporting lines and supporting departments, such as a test expertise centre for example, have been omitted.

Explanation of the examples:

- Example a
  The test manager is completely independent of both the project manager and the subproject realization lead, and works at the same level as the project manager.
- Example b
  The test manager is dependent on the project manager, but independent of the subproject realization lead, and works at the same level as the subproject realization lead.
- Example c
  The test manager is dependent on the subproject realization lead.

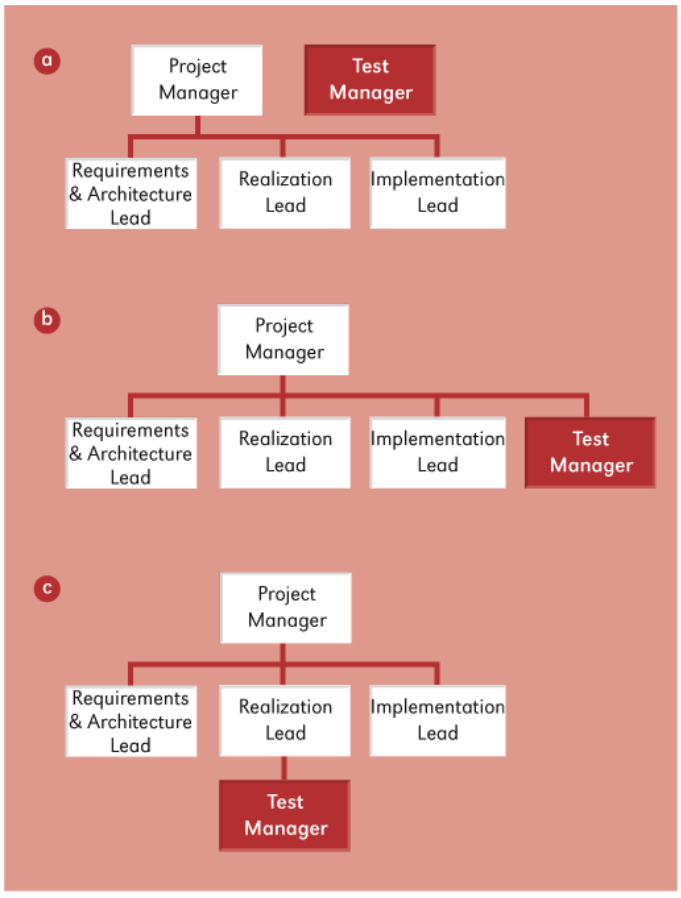


Figure 2. Examples of positions of the test manager in projects

2.3 Building Block 3: Assignment

In general: no job without an assignment. This also applies to a job such as establishing the quality of a product. On a high level, one could say that the assignment must make clear

to the stakeholders just what the aim, tasks, responsibilities and authorizations of the job are.

Assignments come in different flavors. For instance, an assignment in a traditional development environment will differ from an assignment in an Agile development environment and an assignment in a low-risk product environment will differ from an assignment in a high-risk environment. Does this mean that there are no general activities that can be undertaken to establish an assignment? No, on the contrary. The form of the end result of these activities – in a plan of many pages or only a sketch on a white board – may vary, but generic activities can certainly be identified.

A **non-exhaustive** overview of possible questions you have to answer in order to clarify the assignment:

1. Who is the client? The client is the person giving the assignment for the job. It could be a business manager, project manager, steering committee, scrum team, etc.
2. Who is responsible for executing the assignment? It could be a scrum team, test manager, project manager, a third party, etc.
3. What exactly is the job? Think of determining the quality of the product, covering (product) risks, meeting predefined test goals, etc.
4. Who are the acceptors? Often the person executing the assignment is not the one who accepts the product. So who does? It could be a business manager, a product owner, a group of stakeholders, a scrum team, operations, etc. Please note that the client does not have to be the – only – acceptor.
5. What are the acceptance criteria?
6. Determine the scope of the assignment. Determine not only what is within the scope, but also what is outside the scope of the assignment. Determine, for instance, the boundaries of the system (which interfaces with adjacent systems are within / out of the scope), whether administrative organization procedures are in scope or not, where applicable which test levels (e.g., unit test, system test, acceptance test) are within the scope and which are executed by other parties, etc.
7. Which preconditions must be met? Preconditions describe the requirements imposed on the assignment by other parties within the assignment. Such requirements may be: 'operate within the existing quality, risk and/or testing policy', 'meet previously quantified aspects like the risks to be covered, results or quality to be achieved', the time limit in a business case or other planning schedules, etc.
8. Which information will we share with the stakeholders? This can be done in many ways. Agreements are often made with the client and possible other stakeholders about reporting. Often risk and quality reports are desired. Reporting can be done at a regular or *ad hoc* basis and/or at the end of a project. Many organizations have standard forms in which to report.

2.4 Building Block 4: Test Organization

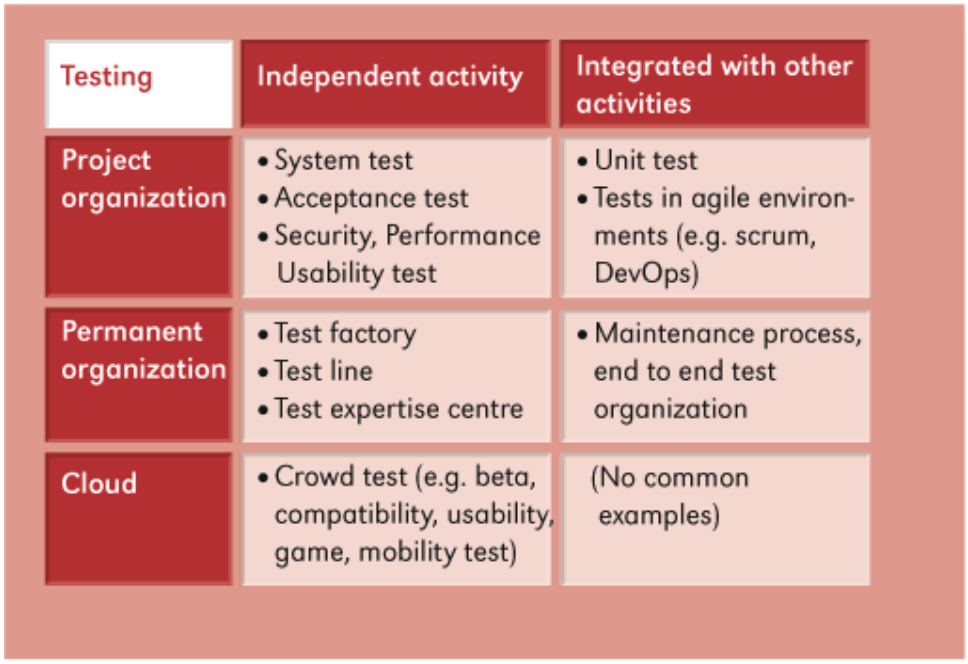
The management and execution of testing can be implemented in many forms. After all, everyone is responsible for quality, and almost everyone is involved in some form of testing. Therefore it is impossible to determine one particular, preferred organizational setup for testing. In general, the structure of the test organization should resemble that of the associated process of system development. In many cases, this means the project organization. If there is to be frequent (re)testing in combination with scarce (test) knowledge, the permanent test organization (e.g., line or staff organization) becomes a candidate.

Organizational implementations for testing

The most significant organizational forms are briefly mentioned below, along with a few examples. For the organization of testing activities, the possibilities can largely be defined as follows:

- Testing as an **independent activity** *or* **integrated with other activities**
- Testing placed within a **project**, a **permanent (test) organization** *or* in the **cloud**

These choices depend on the test variety (refer to the 'Test Varieties' building block), the project and the organization. For possible organizational implementations, see figure 3.



Testing	Independent activity	Integrated with other activities
Project organization	• System test • Acceptance test • Security, Performance Usability test	• Unit test • Tests in agile environments (e.g. scrum, DevOps)
Permanent organization	• Test factory • Test line • Test expertise centre	• Maintenance process, end to end test organization
Cloud	• Crowd test (e.g. beta, compatibility, usability, game, mobility test)	(No common examples)

Figure 3. Organizational implementations with examples

The descriptions below are explicitly meant as a general indication; there are often exceptions in practice:

- **Testing as an independent activity in a project**
  Within the project, a team is responsible for organizing and executing the test. The testers within the team usually have a lot of test knowledge, together with – depending on the test variety – a mix of system, domain and organizational knowledge.
- **Testing integrated within a project**
  In traditional development projects, test activities involving unit testing are integrated into the development process. In Agile projects, testers, users, designers and developers work in the same team. There are often several teams (e.g., scrum or DevOps teams) in action. Testing is a role within this team and the team as a whole is responsible for executing the test. Besides profound testing knowledge, each team member with a testing role, *as a rule*, also has much domain and technical knowledge of the system and architecture (refer to building block 18 – integrated test organization).

- **Testing as an independent permanent organization**
  A separate department or organization has testing – both the organization and execution – as its primary task. Projects or other line departments issue a certain test instruction to this department/organization. Test knowledge is predominant. (See building block 13 – permanent test organization – for two common types of organization).
- **Testing integrated in the line organization**
  Within a development or system management department, the role of tester is often combined with other roles (e.g., with the role of functional maintenance). The tester in this organizational form often possesses considerable system and/or organizational knowledge.
- **Testing as an independent activity in the cloud**
  Crowdsourced testing is an emerging trend in software testing which exploits the benefits, effectiveness and efficiency of crowdsourcing and the cloud platform. It differs from traditional testing methods in that the testing is carried out by a number of different testers from different places, and not by hired consultants and professionals. Often these testers are very skilled, but the quality of the testing work can vary since you never know who will be involved in a particular testing task.

2.5 Building Block 5: Test Plan

Failing to plan is planning to fail. So we do need a test plan – and what should be in it? Project plans are often crafted but sometimes hardly read in real-life practice. So should there be a separate test plan? There are questions about testing that pop up in all situations, and need to be addressed.

Think of:

- What do we need to test?
- Who will test what at what moment?
- How will we test?
- How much time will the testing take?
- When is the test ready?
- How can we organize and manage the testing?
- What kind of test products do we need to deliver, and can we manage them?
- What kind of test environment do we need?

Of course there could be more questions, but these are the most important.

Elaborating these topics guides our thinking on important matters and forces us to confront the challenges that await us. The results of the elaboration (which preferably includes several options) may be established in a test plan, depending on the applied development method, (product) risks involved, trust, purpose, regulations, responsibilities, etc. Since a plan is primarily a means of communication between the various stakeholders, the following questions can be posed:

- Why are we writing this in the plan: does the team need this in order to carry out our work?
- For whom are we writing this in the plan: do the stakeholders have to know this?

The answer is situation-specific, of course, but if the team and the stakeholders respond negatively to both questions, writing these topics in the plan would appear to be unnecessary.

So, what could a plan look like? In practice, this varies from a sketch on a whiteboard to test aspects integrated in project plans (or sprint plans) to documents with many pages. Often there are Master Test Plans (gearing the various test varieties to one another) and Detailed Test Plans (elaborated test plans on test variety level). And sometimes there is no visible plan at all. As when, for instance, one is working in and as a team and the members understand each other blindly.

2.6 Building Block 6: Product risk analysis

Projects should follow an integral testing and quality policy, for quality is a mindset, not a feature. Quality as such ought to be an integral part of project management. Looking at project governance, several aspects play an important role: costs, risks, time and benefits (also referred to as 'results' or [business] value).

Although all of these aspects are important, it is worth emphasizing the aspect of risk, especially product risk, as this may help as a steering mechanism. It may help you find the balance between 'building the right thing', 'building the thing right', and 'building it fast'.

No project has unlimited time, money and resources for assessing the quality of the product. Such constraints in terms of time, money and resources represent constraints on the result to be achieved and therefore often mean reduced possibilities to assess the product risks. As such, it is important to achieve a well-considered balance between the investment in money and time on the one hand, and the results to be achieved and the risks covered on the other. The result of the product risk analysis provides the justification for this balance. Based on insight resulting from the product-risk analysis, high-risk products can be covered more intensively than those representing a lower risk. Be aware that risks and ways to cover these risks are directly related to the acceptance criteria (see the 'Assignment' building block). These acceptance criteria are available in various forms: as a section in a test plan, in the confirmation part of a story card, in a definition of done, etc.

There are many approaches to the way risks are determined. However, in general, one could say: these approaches involve analyzing the product to be assessed with the aim of achieving a joint view – *for* and *with* all stakeholders – of (the properties of) the product to be assessed, in terms of higher and lower risk levels. This should be done in such a way that appropriate measures can be assigned to this view.

Exactly which measures should be taken to cover these analyzed risks is decided when determining the quality strategy. This means that the right quality is designed and built in, not tested in! This means that everyone should embrace risk and quality thinking right from the beginning of the project. When a requirement is described (e.g., user story, use case), for instance, the staff should start thinking about possible risks and how to cover these, by executing an inspection of the requirements (such as a Fagan inspection for example). Or when the system architecture is (being) defined, they should rethink the possible risks and how to cover these, by means of a proof of concept. A last example of risk coverage is to assign specific coverage types in order to cover identified (product) risks. Read more about risk coverage, and the measures that can be taken as part of the test strategy in the 'Test Strategy' Building Blocks. But first things first. How do you determine the risk level anyway? A good starting place to find more approaches is tmap.net. There you will find

approaches such as: TMap's Product Risk Analysis, PRISMA®[1], PRIMA®[2] and Risk Poker in scrum. Although there are many useful approaches, the basis often proves to be surprisingly similar. Just look at the definition of product risk, which will differ in wording, but not in meaning:

A product risk is the chance that the product will fail in relation to the expected damage if it does fail:

Product risk = Chance of failure x Damage

Sometimes 'chance of failure' is referred to as 'likelihood of occurrence' and 'damage' as 'impact'. It really doesn't matter because the result will ultimately be similar, if not the same.

Three types of risk can be distinguished. When the primary effect of the potential problem relates to product quality, potential problems are referred to as 'quality risks' or 'product risks'. When the primary effect of the potential problem is on the process of the organization, it is referred to as a 'process risk'. When the primary effect of the potential problem is on project success, potential problems are referred to as 'project risks' or 'planning risks'.

The above is about identifying risks and risk coverage. Sometimes people look upon things in a different way. They try to answer questions such as: What is the importance of a particular product? Or what is the urgency of a particular product?

This is important, of course. The product delivers most added value if the product is realized at the right time. And if this is not done correctly it may have a hugely negative impact on the success of the project (which is an example of a planning / project risk). This can also be combined with the product-risk analysis. Examples of this kind of integrated approach are 'Product Risk and Benefit' (PRBA) analysis or 'Risk and Value' analysis.

2.7 Building Block 7: Test strategy

The assignment, the product risk analysis together with the test strategy form the basis of virtually all test activities (refer to the 'Assignment' and 'Product Risk Analysis' Building Blocks). The product risk analysis contains the legitimacy concerning what must be tested and which risks are inherent in the process. The test strategy defines which of these ought to be covered and how. This may influence the priorities in a project, in the sense of the greater the risk, the higher the priority (and desired coverage), for instance. Decisions involving what should or should not be done, with respect to time, costs and benefits (also referred to as results or business value) may also be influenced. An often-used definition for test strategy is the following:

"The distribution of the test effort and coverage over the products to be tested aimed at finding the most important defects as early and cheaply as possible."

This definition is closely related to covering risks. But what about the other project aspects such as costs, time and benefits? After all, each project emphasizes one or several of these aspects (refer to 'Assignment'). The emphasis must be translated into specific choices

[1] PRISMA is a registered trademark of Improve Quality Services

[2] PRIMA is a registered trademark of Valori

in the test strategy. An emphatic choice for one of the aspects often has an impact on the other aspects. If there is a maximum budget, for example, maybe not all identified risks can be allocated the desired coverage, which could result in the fact that the acceptors have to tolerate a certain residual risk when releasing the item into production. Or, with respect to a mission and safety critical system with a sky-high business value, for example, the risks must be covered thoroughly. However, this may involve additional time and cost. Besides all this, the following principles are often applied as well:

- Defects must be found as close to the defect injection point as possible (fewer repair costs, quick learning curve).
- The bigger the risk, the more intense the test.
- No risk, no test.

Please remember that these are *principles*. 'No risk, no test', for instance, is something that will not occur in practice when building a piece of software. There will always be some risk involved, or the piece of software isn't worth developing in the first place.

All these considerations can be captured in a strategy table. Just as there are many approaches for carrying out a product risk analysis, there are also many kinds of test strategy tables. The following list of test strategy aspects is in no way complete or mandatory. It merely provides a start-up list from which aspects can be removed or changed or added. Aspects addressed in a test strategy table could be (refer to figure 4 'Test strategy table'):

- Risk
  Risk is a generic term, but may include: test risks, product risks, project risks or planning risks.
- Risk Level
  The level of risk as decided by all stakeholders during a product risk analysis.
- When
  One of the test strategy principles runs as follows: 'The earlier a defect can be found, the better – if useful.' So it is important to decide when a quality measure must be executed.
- Location/ By Whom
  This clearly shows, where and by whom the responsibility lies for the execution of the quality measure(s).
- Test Variety
  A test variety represents a certain need for testing, no matter how this is organized. Other terms may be used such as test level, test type, etc.
- (Quality) Measure
  In order to cover a specific risk, one or more quality measures are assigned to cover this risk. While assigning the quality measure(s), the level of risk is explicitly taken into account.

Of course, you should adapt the table completely to your own situation!

Risk (regarding:)	Risk Level	When	Location/ By Whom	Test Variety	(Quality) Measure
Architecture	High	Design Phase	Team	n.a.	Proof of Concept
Test Knowledge	Low	Project Startup	Sogeti	n.a.	Test into Course
User Story x	High	DesignPhase	Team	Evaluation	Inspection
		Test Phase	Team	Acceptance Test	CT: Path
Performance	Medium	Realization Phase	Third Party	Performance Test	CT: Load Profile
Change Request y	Low	Design Phase	Team	Evaluation	Review
		Test Phase	Team	Acceptance Test	CT: Data Row
Requirement z	High	Design Phase	Team	Evaluation	Inspection
		Test Phase	Team	Exploratory Test	n.a.
...	...				

Figure 4. Test strategy table (CT: Coverage Type)

2.8 Building Block 8: Performance testing

People often use the statement 'No Risk, No Test'. For the performance of IT systems, the 'No Risk' situation simply doesn't exist anymore. Performance optimization is critical when the user-experience in even the most trivial system can be impacted by bad performance. This optimization becomes even more important with technology integrating in all aspects of our professional and personal life (i.e., cloud-based computing, mobile solutions and the Internet of Things).

Perhaps the most important aspect of performance testing is the organizational aspect. The IT organization (in the form of a Performance Test Expertise Center for instance) must be able to support organization-wide changes in Design, Develop, Testing and Maintenance practices with regard to system performance. This requires the capability to implement tools and methodologies that meet project-specific needs and requirements (Agile, non-Agile and maintenance). In Agile specifically, performance testing must be able to support work processes in sprints as well as across sprints or teams. This performance-testing effort can be supported via a structured performance-testing approach, providing easy access to tools and resourcing from a specialist resource pool.

Performance testing is possible in different test varieties, with a big difference in the type of tools used, skill sets needed and, most importantly, the intensity of testing and analysis. The most important Performance Testing varieties with some (but not all) of the activities required are shown:

Design for Performance Testing

For a long time, performance testing's main concern was to gather requirements and expand the performance aspects of those requirements. In many performance test activities, this at least resulted in knowing when performance was (extremely) bad. Design for Performance puts the focus on designing for good performance, and providing SMART requirements as input for validating that performance. The Performance Testing effort is

not actively involved in all aspects of Design for Performance, but provides the framework for capturing performance-related requirements from an early stage.

Activities:

- Making sure to include click-path or workflow descriptions into UI and application design: This provides useful and SMART requirements for performance testing in all application lifecycle stages
- Attention for performance aspects: From a Product Risk Analysis that focuses on specific aspects of the system (instead of just "Performance is a high priority") to end-user involvement on click-path design and documenting expected usage levels (this is also an important Product Owner responsibility)
- Designing for performance testability: Without compromising on quality, design choices on everything from database to security aspects can facilitate the optimal use of available performance test tooling (or signal the need for additional tooling).

Develop for Performance Testing

At the most technical level a project must look at performance testing at a component (unit testing) level, as well as a system (integration testing) level. This results in working with - platform-specific best practices in design and development. With different design choices (and performance consequences) for anything from ERP-based systems to portal or mobile solutions, there is no one-size-fits-all solution. Constant vigilance is required to keep up with new versions of development frameworks and the resulting performance impacts (both good and bad). This 'Develop for Performance (Testing)' approach is then validated for the first time in testing specific components (web services/ database access etc.) during the Unit/Integration testing phase.

Activities:

- Develop for performance: Use of best practices (market as well as company specific) with a focus on clearly defined application components (i.e., Service Layer design patterns)
- Platform (including database) specific tooling and training: Test-driven development. If applicable, a performance test must be designed and executed as part of unit and integration testing
- Implementing specifically designed stubs or simulation software for not yet available system components.

Acceptance Performance Testing

This is considered the traditional approach to performance testing. Load and Iteration Models are created, based on user profiles and click-paths through the application. These models ought to match the expected load and mix of usage patterns for a large group of end-users.

Activities:

- Deploying and using Performance Test Tooling

- Network traffic Capture and Playback tooling: Designing, building and maintaining the (production-like) test environment and using the available tools to create individual performance test scripts
- Multi-load generator controllers: Designing and implementing the performance scenario (mix of test scripts) to simulate the load on the system, as defined in the requirements.

End-to-End Performance testing

As part of a continuous attention to performance optimization, the impact of different applications on the total landscape must be tested. A permanent test organization (i.e., Performance Testing Expertise Center) can combine and re-use scripts and scenarios for multiple applications into an End-to-End Performance Test.

Activities:

- Deploying and using Performance Test Tooling as defined in Acceptance Performance Testing
- Test Lab set-up: Design and maintain a lab with enough capacity or flexibility to run multiple applications in a test scenario (additional skills needed in network components, virtual machine management, network experience, etc.)
- Environment monitoring: Running and maintaining similar tooling as that used in the production environment, with the performance test tooling able to tie into those monitoring results.

Production Performance Monitoring

Performance monitoring in the production environment has a number of different goals. The primary goal is to safeguard business processes and provide early warnings of performance degradation. This is done by monitoring available resources throughout the IT infrastructure. By running performance test scripts (for a very limited number of simulated users) in the production environment, the end-user performance experience can also be monitored.

A secondary goal for running test scripts and monitoring results in production is to provide feedback to earlier performance-testing levels. Are the designed tests still an accurate representation of user behavior? This prevents the occurrence of a situation in which a test and monitoring setup no longer represents real-life usage. When more and more traffic is being generated from mobile devices, the resulting load has to run in parallel alongside traditional (PC-based) browser usage in all varieties of performance testing.

Activities:

- Multi-load generator controllers: running low-impact test scenarios (perhaps outside peak business hours) that provide continuous performance results
- Environment monitoring: Running and maintaining monitoring tools and providing reports/results that can be compared to current test results in earlier stages of the application lifecycle.

2.9 Building Block 9: Test approaches

When you have made choices about what needs to be tested and just how thoroughly specific parts need to be tested, the next choice is how to actually test them. There are two approaches to performing tests: experience-based and coverage-based.

Experience-based

Experience-based testing leaves the tester free to design test cases in advance or to create them on the spot during the test execution. These tests are based on the tester's skills, intuition and experience. Part of Experience-based testing may be considered:

- Checklist-based: The experienced tester uses a high-level list of items to be noted, checked, or remembered, or a set of rules or criteria against which a product has to be verified.
- Error Guessing: Based on the tester's experience, he goes in search of defect-sensitive spots in the system and devises suitable test cases for these.
- Exploratory testing: The simultaneous learning, designing and executing of tests; in other words, every form of testing in which the tester designs his tests during the test execution and the information obtained is reused to design new and improved test cases. Exploratory testing can be very well applied with the use of coverage types.

Coverage-based

Coverage-based is a way to derive and select test situations based on an analysis of the test basis, applying selected coverage types in order to achieve a desired coverage. Coverage has everything to do with the wish to efficiently and effectively gather information about quality and risks, and to find the greatest possible number of defects, with the fewest possible test cases, aimed at specific aspects of the test object. A coverage type focuses on achieving a specific coverage to detect specific types of defect.

There are roughly four groups of coverage types.

1. Process: Processes can be identified at several levels. There are algorithms of control flows and business processes. Coverage types such as paths, statement coverage, and state transitions can be used to test (variations in) these processes.
2. Conditions: Within almost every system, there are decision points where the system behavior can go in different directions, depending on the outcome of such a decision point. Variations of these conditions and their outcomes can be tested using coverage types such as decision coverage, modified condition / decision coverage, and multiple condition coverage.
3. Data: Data are created and end when they are removed. In between, the data are used by updating them or consulting them. This lifecycle of data can be tested, as well as combinations of input data, and the attributes of input or output data. Boundary values, CRUD, Data flows and Syntax are examples of coverage types in this context.
4. Appearance: The way a system operates, how it performs, what its appearance should be, is often described in terms of non-functional requirements. Within this group, we find coverage types such as operational and load profiles and presentation.

2.10 Building Block 10: Crowd-testing

Crowd-testing, or crowdsourced testing, is when a virtual group of testers throughout the world (a crowd) is involved in testing, rather than only a traditionally managed test team at a single location.

'Crowd-testing' has its roots in 'crowd-sourcing'. The key to the success of crowd sourcing is that a lot more ideas can be generated within a larger group of people, and those ideas can influence and support each other. Current Cloud and Web 2.0 technologies enable the sharing of ideas within large groups. This same advantage applies to crowd-sourced testing: many testers can find plenty of places to look for bugs, and the bugs that one tester finds can influence other testers to look for similar bugs.

Crowd-testing has a second advantage. It is not just the ideas and experience of the different members that are more diverse, but also the different hardware and software configurations that the software under scrutiny is tested against. Crowd-testing enables a test where software can be tested on a large number of devices, browsers and operating systems, where all tests are executed in parallel, minimizing the throughput time of the tests.

Managing a crowd of testers entails a different set of challenges than managing a traditional test team. The following are some of the considerations to take into account:

Testers or end users

Do you want your crowd to go actively looking for bugs or do you want them to use the software as they would in real life? In the first case, pick experienced test professionals as crowd-testers. In the second case, pick typical end-users.

Organize your own test crowd or use a crowd-testing company

In some cases it is not possible to release the software beyond the bounds of your own company. Financial institutions have an issue with releasing software to a large crowd of relatively unknown people. If this is the case, you can organize a crowd test (or more exactly: a closed beta-test) among your own employees. However, if there are no constrictions on releasing a test version of your software outside the company, there are numerous companies available with a global crowd of testers that can amount to hundreds of thousands.

Rewarding the crowd

There are different ways to manage the crowd. One thing to consider is: how do I reward my testers? If you are managing a closed beta within a company, rewards are often for executing the whole test. For instance, if your company members test a new kind of device, they can keep the device after the test.

If you are managing a public cloud, there may be other ways to reward testers. Two possibilities are often used:

- Payment for each bug found
- Payment for each test case executed.

A disadvantage of rewarding the crowd for each bug is that, with a primary focus on finding bugs, it may be difficult to get a good picture of the overall quality of the system under test.

Disadvantages of crowd testing

Crowd-testing offers some clear advantages as stated above. However, there are some disadvantages that limit the use of crowd-testing as it is currently developing.

- Confidentiality: the larger the crowd and the more it is outside the sphere of influence, the harder it becomes to manage confidentiality.
- Knowledge: not every application is suited for crowd-testing from a knowledge perspective – many applications are used within a company and specific knowledge of company products and processes is required to operate the software.
- Testers who are paid 'by the bug' will often look for easy-to-find bugs instead of looking for the most critical ones.
- Ensuring total test coverage can be difficult with crowd-testing.

2.11 Building Block 11: Test varieties

When organizing testing, the test manager adhering to the traditional view on testing had to structure the testing activities in a hierarchical way, based on quality characteristics. But a test manager often distinguished various stages too. Defined terms such as Test Level, Test Type, Test Phase and Test Stage were often used.

In today's view on testing, the people involved in testing are hesitant to use the word 'Test Level' since it seems to imply that various groups, based on various hierarchical responsibilities, will perform various testing tasks without any interaction between these test levels.

Moreover, many testers have often struggled to distinguish between Test Levels and Test Types. And a Test Stage – is that identical to a Test Level or not?

What should be our focus when organizing testing?

All testing activities must collectively cover all important areas and aspects of the system under test: that is the main objective.

To cope with the confusion around how to distinguish testing tasks, we introduce the term Test Variety.

The term Test Variety aims at making all stakeholders aware that there will always be different needs for testing, and therefore different test varieties will have to be organized. Whether these are organized separately or combined depends on the situation.

There may be many reasons for having different test varieties. For example, there are different stakeholders who ought to be involved: programmers have a different focus in their testing than business representatives do. This is often related to responsibility and accountability for testing activities. The quality characteristics that have to be addressed form another reason for distinguishing test varieties. Maintainability for example, demands totally different testing activities than usability does.

Traditionally, different aspects were separately approached as a group of testing activities that had been brought together in a test level. Many people know the 'functional acceptance test', whose name already indicates that testing was not complete because it obviously didn't focus on non-functional aspects. In the new view, functional and non-functional testing can be seen as test varieties. Depending on the circumstances, such as the application lifecycle model that is used, these test varieties are organized either together or separately. The main concern is that all relevant test varieties are carried out one way or another.

Inexperienced Agile teams tend to focus their testing efforts on 'unit testing'; that is, testing whether or not computer programs meet the technical needs. This is definitively one of the important test varieties. But another important aspect is to validate whether or not the business goals have been met, the 'acceptance testing', which may be done by the product owner in an Agile team. Not all Agile teams realize that this test variety is equally important. In practice, these varieties of testing must be done by the Agile team in the same iteration, so the test varieties in this example can be considered as having been organized together, even though different team members may work on different test varieties. Of course, the Agile team will also distinguish additional test varieties, such as performance testing and security testing, which might also be done during the iteration.

However, especially in a larger organization, the people involved also will see the need for test varieties that cannot be done by a single team within their iteration, but have to be organized separately instead, such as an end-to-end test.

If you have ever taken part in a discussion on whether end-to-end testing is a test level or a test type, you will recognize that this doesn't actually matter, as long as the testing activities related to the end-to-end business process are carried out properly.

So let's use the term Test Variety, to make everybody involved aware of the fact that there are different points of view towards testing activities, and we can make sure that the interests of all stakeholders will be covered by addressing these in a well-considered way.

2.12 Building Block 12: Test Manager in Agile environments

Test managers tend to be quite nervous about Agile. As the focus of a testing team switches to collaboration on products and projects, rather than testing being an isolated phase or service, it may feel like the need for a test manager disappears. Because testers should be communicating their progress directly within their project teams, providing their estimates as part of an Agile methodology and using just-in-time test planning, there would seem to be no need for a test manager who acts as an intermediary or overseer at a project level. But what about the other test-management activities? How does Agile take care of that? Let's look into the scrum example below:

> *There are three roles in a scrum team: the product owner, the scrum master and the team members (or developers). The team is self-organizing and multidisciplinary, without managers. There is no room for test managers in this type of team. Testing is a role that every team member should be able to execute. The fact that 'test manager' has not been adopted as a function in a scrum team does not mean that the test management activities should not be executed. On the contrary, these remain unfailingly important. But they may be executed by any random team member with the appropriate expertise and skills. Nevertheless, it is advisable to have a professional tester in the team, to guarantee available test*

expertise. This tester (might be a former test manager) has knowledge of the execution of a risk analysis, the execution of evaluations, test design techniques, the formulation and execution of test cases, test automation, etc. But this does not mean that all test activities must be executed by this tester. Other team members may be requested to provide support in the creation and execution of the test cases, for example. In such a situation, the professional tester can act as coach. If a team cannot guarantee sufficient test expertise, it may be an option to allow a test manager from outside the team to support and coach the tester(s).

But in general, in Agile environments, one could see the test-manager role as evolving to a higher-level position that includes or concerns:

- In sprint zero: advisor to the team – how to cope with responsibility for quality?
- Facilitation of inter-team communication across many Agile projects within an organization
- Presenting an aggregate view of testing utilization to high-level management
- Personal support, mentoring, and professional development for testers (e.g., as a line manager)
- Being an escalation point for testers
- Budgeting or forecasting for testing as a service (dependent on organizational process – testing as a service must be used)
- Being involved in scrum-of-scrum meetings
- Providing advice regarding quality
- Functioning as a stakeholder for the product owner
- Combining with the scrum-master role.

2.13 Building Block 13: Permanent test organization

Two types of permanent test organization are common in actual practice. These are (see figure 5):

- The permanent test organization as a test expertise centre (TEC)
- The permanent test organization as a test factory (TF) or test line (TL)

The two differ in, among other things, the services they offer and their responsibilities in this respect. The tec (often implemented as a "staff organization") is mainly a supplying and advisory organization that takes on an 'obligation of effort' at most when providing services. For instance, it may supply testers or test managers for a project or even for another line organization within the company. Or offer advice on a test method of operation or test tool to be used (e.g., to a scrum or DevOps team). The activities are always executed under the responsibility of the project.

The TF or TL accepts an 'obligation to deliver results' for many of its services. The process can be compared with a factory with permanent personnel (testers), machinery (infrastructure), standardized work procedures, etc. Different clients (departments, projects, systems) can outsource their complete test assignments to this type of test organization that is organized as a 'line organization'.

The term Test Competence center of excellence also pops up often, when talking about a permanent test organization. This can be any of the mentioned structures.

Both test organizations make a distinction, based on demand frequency in the test services. The test service is approached from a different perspective for incidental requests ('set up a test environment') than for structural requests ('test releases').

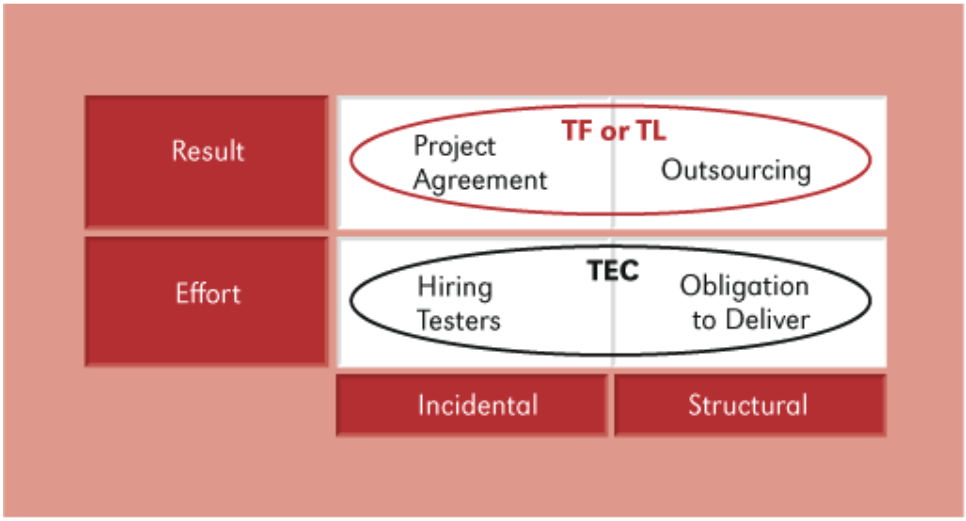


Figure 5. Two common types of permanent test organizations

2.14 Building Block 14: Model-based testing

Creating test cases in any development method is at the core of testing and can be very time consuming, especially when conducted manually. Also, test case development can be prone to interpretation errors when the test base – be it requirements, design documents or any other artifact – is ambiguous.

Model Based Testing (MBT) ranges from full-blown test automation in which test cases from models are created and executed all in one go by an MBT suite via Model-Based Test Design (MBTd), aiming at shortening test case development lead time to Model-Based Review (MBR). In turn, this aims at reducing test base ambiguity but without delivering actual test cases.

Model-Based Review

In MBR, models are means to an end, the end being verifying that the source of the test cases is clear and complete. The tester composes one or more models so that end users, analysts, designers etc. can verify the tester's understanding of the subject. The source can be tangible documents, but also 'in the heads of anyone'.

The basic compelling idea behind MBR is that models are unambiguous by nature, so flaws such as incompleteness, inconsistency and incorrectness catch the eye more easily.

Models are also a limited view on reality, so often several models need to be composed to represent a complete picture of what's in the design artifacts or 'in the heads of' those involved. For example, a process is best represented by a flow diagram, but a 'Yes/No' decision in that process might be subject to several basic 'Yes/No' conditions. These conditions could be modeled individually and explicitly in the flow diagram, but the model of preference for conditions is the decision table or pseudo code.

Model-Based Test Design

MBTd builds on the unambiguity of models: they can be automatically interpreted and converted to test cases. The model's completeness and level of detail determines the ability to derive physical test cases for automated test execution, logical test cases for manual testing, or anything in between. The architecture of the system under test is also an important factor in the feasibility of the end result: system-testing a SOA-based application is a better candidate for automatically executed physical test cases than End-to-End testing that involves many systems beyond the control of the tester.

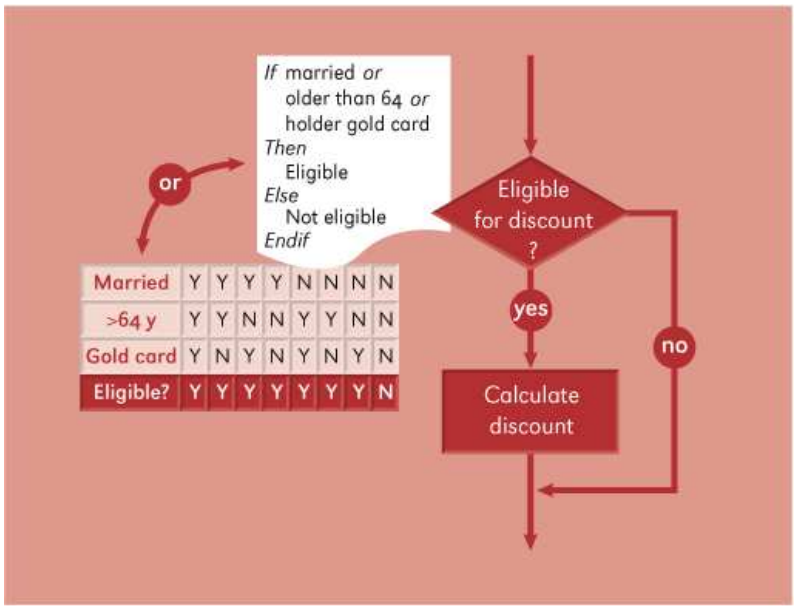


Figure 6. Model Based Test design

The basis for MBTd can be either design models or 'test models' from MBR: (re)using design models is often the quickest and easiest way to implement MBTd. There is one aspect, however, that deserves special attention: do the design models contain sufficient detail to satisfy the tester's test goals? This is not always the case: if, for instance, the test goal is to verify adherence to design standards, it is very unlikely that the design models explicitly model these standards! Complementing existing models with test specific details might prove to be more labor-intensive than formulating test models from scratch.

Full-blown automated Model-Based Testing

At its optimum, MBT reduces the testing effort to creating and/or reviewing models, after which one push of a button suffices to create and execute the test. There are several suites that deliver this capability, but a multiple-step approach can also be viable, employing different tools for different steps, making a staged implementation of MBT possible. One reason for a staged approach is the opportunity to re-use the installed base of tools and automation frameworks.

A very nice 'side effect' of MBR is the gradual transference of pure test models to all-purpose models, used in analysis, design and test alike, because designers assume ownership after reviewing the models. So MBT integrates in a bi-directional way: testers use design models for MBTd, and designers assume ownership of MBR test models.

Perhaps the greatest benefit of MBT lies in maintenance: adjusting a (test or design) model and then regenerating tens or even hundreds of test cases with the push of a button can never be equaled by manual test case maintenance: not in terms of lead time, not in terms of cost and not in terms of quality!

MBR and MBTd each bring their own individual benefits, but the combination of the two is the strongest application, at best eliminating interpretations errors and averting manual test execution.

2.15 Building Block 15: Quality policy

In general, companies that make structural use of testing do have a test policy.

Companies that consider quality to be of structural value have a quality policy. 'Policy' is used here as the overarching term. Other terms used in this context are 'mission', 'vision', 'strategy'.

A quality policy includes choices made by management that are generally applicable to operating activities. Sometimes it has the form of a formal and certified quality system. That, too, is a management choice. ISO9000 is a well-known standard for quality certification.

In the main, a quality policy ensures that an organization, product or service is consistent and is focused not only on product and service quality, but also on the means to achieve it.

What are the constituents of a quality policy?

A quality policy is always based on the company's strategy. It commonly includes subjects like: vision on quality, objectives, scope and quality principles (e.g., about customer focus, leadership, people, a systematic approach through processes, used quality standards or models, continuous improvement, etc).

If it concerns IT projects it will incorporate subjects such as: ambition level on quality, continuous improvement, methods used, common tooling, how quality and test expertise is organized. In general: all the choices that are cross-projects.

When do you need a quality policy?

That decision is up to top management, but generally it is needed to ensure that quality aspects are treated in the same way throughout the entire company, in a way that reflects the company's values. When a policy is needed, then the redaction, application and control is typically assigned to a QA staff department.

How does one create and maintain a policy?

The following figure shows the processes involved in creating, applying and maintaining a quality policy in a project environment, related to the change process.

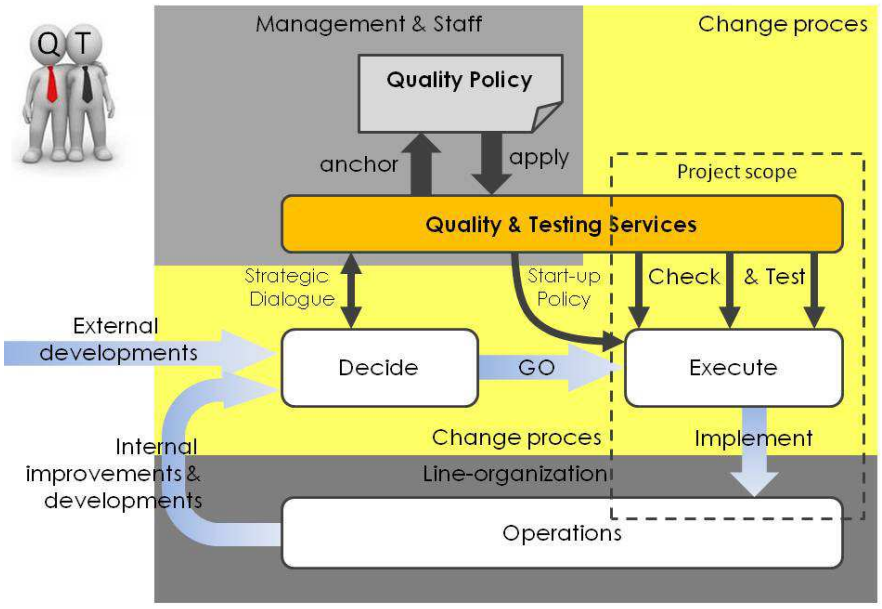


Figure 7. Quality policy process in a project environment

The change process is central, essentially consisting of two steps: decide on how to react to developments and execute the decisions. The figure shows quality and testing services to support this process. Usually assigned to a QA staff department, these services consist essentially of three parts:

- The decisions process is supported from the quality perspective. The existing policy is applied, adapted or extended in a strategic dialogue with the deciding management.
- When the decision has been taken to start a project, the project is supported by a start-up policy.
- During the execution phase, the project is monitored, by means of tests, to check the quality.

How does one ensure that an it project actually applies the policy?

After the decision on how to react to developments has been taken, the necessary changes are determined and it projects are initiated. When a project is in the start-up phase, an extract can be made from the overall policy, concerning the aspects that are applicable given the goal of the project: methods, tools, test strategy, etc. In fact, a quality and test plan can be drawn up in conjunction with the project plan. This ensures that project scope, budget and time take sufficient quality and testing efforts into account.

How is a quality policy related to testing?

A quality policy contains measures to check and show the actual quality of anything subjected to the policy. Testing is one of the measures. Testing is an excellent measure to show the actual quality. In TMap, the test strategy defines how testing is addressed and is included in a master test plan. The so-called Generic Test Agreements (GTAS) resemble a

master test plan and sometimes even replace it. A test policy supersedes GTAS that can be related to specific outsourcing situations. A quality policy indicates how testing is used as a means to measure and demonstrate quality.

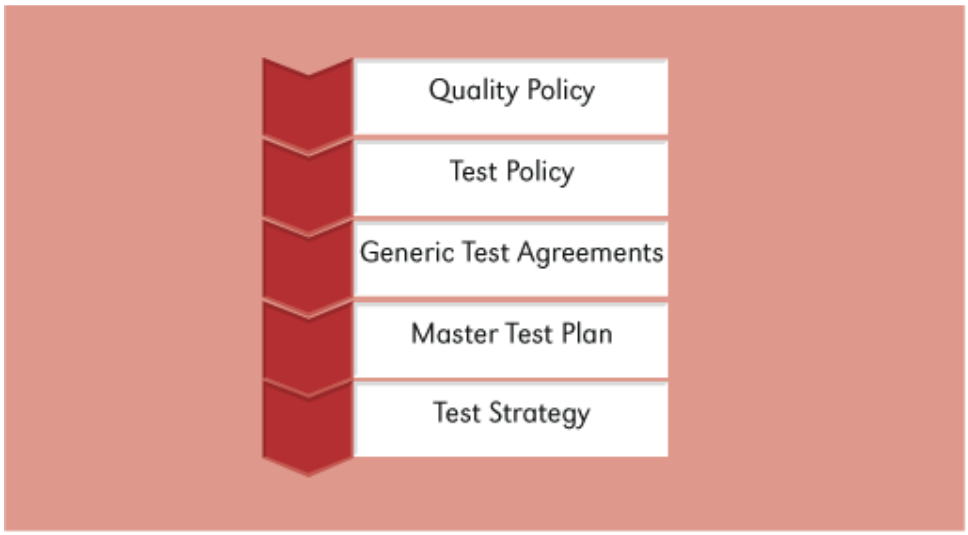


Figure 8. Quality Policy Relation to Test

Differentiating factors in the difference between success and failure when setting up a quality policy may relate to factors such as commitment, knowledge and expertise to guide improvement, the scope of the desired improvement, and adaptation to the corporate culture. To set up a quality strategy, it is important to apply the element of People, including culture and teambuilding. Any improvement (change) takes time to implement and stabilize as accepted practice. Improvements that change the culture take longer, as they have to overcome greater resistance to change. It is easier and often more effective to work within the existing cultural boundaries and to make small improvements (that is Kaizen) than making major changes in one 'big bang'. On the other hand, a 'big bang change' works best when an enterprise faces a crisis and needs to make major changes in order to survive. A well-defined quality policy will take all these factors into account.

2.16 Building Block 16: Using test tools

There are lots of different types of test tools, each with its own purpose. We can classify test tools by stating the testing activities they support:

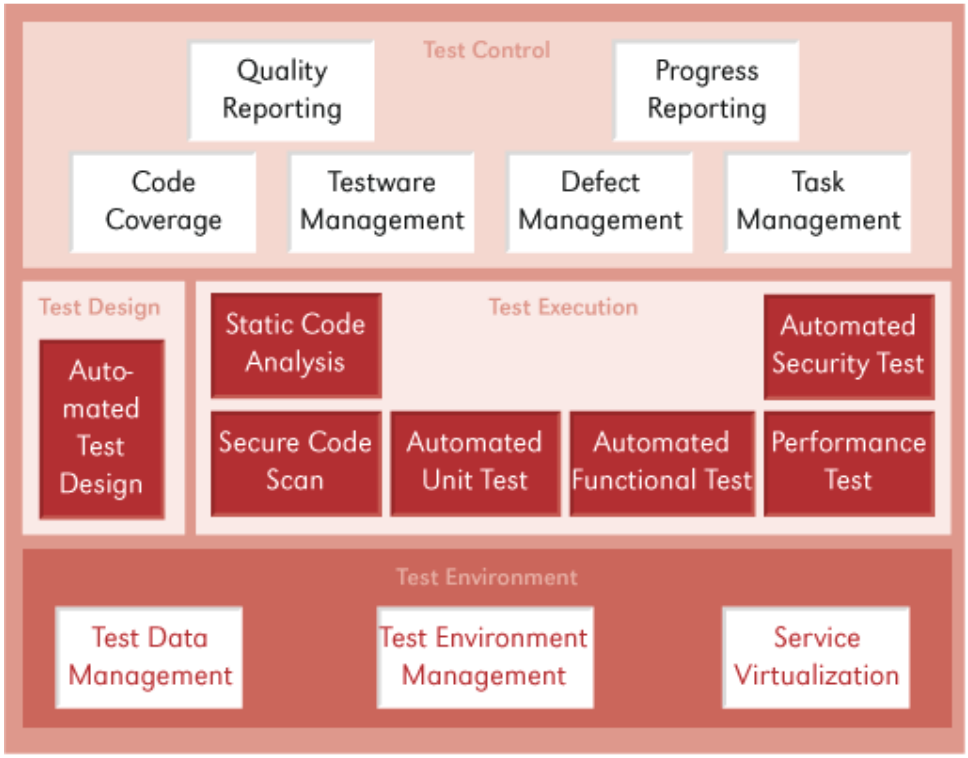


Figure 9. Test Tool Classification

This does not mean that these individual activities are supported by a single test tool. Most test management tools actually feature a combination of testware management, defect management and reporting capabilities for example.

The use of any of these tools is aimed to produce an effect. It's useful to distinguish primary and derived effects. Test execution tools accelerate test execution, so the primary effect is saving time. There is a choice in the derived effects: either reducing test execution time, increasing coverage in the same test execution time, or increasing the number of times the tests are executed. The exception in this category of test tools is a performance test tool, whose primary effect is the ability to execute a performance test; the derived effect is insight in performance and stability.

The other types of test tools have different effects. Test control tools have the primary effects of quality and progress control, test design tools save time, and test environment tools enable control over the preconditions to execute tests.

The primary and derived effects of the most commonly used types of test tools are given below. Multiple derived effects means a choice has to be made which effects will be achieved.

Type of tool	Supported activity	Primary effect	Derived effect
Test management tool	• Testware management • Defect management • Quality reporting • Progress reporting	Control over test products	Control over quality
Code Coverage analysis tool	Code coverage	Insight in test coverage on code level	Control over quality
Model Based Testing tool	Automated test design	Saving time	• Reduce test design time • Increase coverage for test execution
Source code analyzer	Static code analysis	Insight in code quality	Control over quality: • Technical
Unit testing framework	Automated unit tetst	Saving time	• Increase coverage • Test more often
Test execution tool	Automated functional test	Saving time	• Reduce test execution time • Increase coverage • Test more often
Performance test tool	Performance test	Executing load and stress test	Control over quality: • Performance • Stability
Test data management tool	Test data management	Have the right test data within the constraints of privacy legislation/cost for test environments	• Reduce test execution time • Increase coverage for test execution
Service virtualization tool	Service virtualization	Reduce dependency of availability service	• Test earlier • Test in parallel • Test always

Figure 10. Type of tools, application and impact in the short and long term.

Cost reduction is always a derived effect. The remarkable thing is that the main financial benefits of using test tools is not within the test process itself: reducing test time benefits the business by adding business value earlier, improving quality benefits operations by reducing the number of incidents, and finding defects earlier benefits development by decreasing the costs for fixing them.

The effects of individual types of test tools can be increased by combining or integrating them. And even more benefit can be gained by combining or integrating them with other tools used in the application life cycle, such as tools used in the development process for requirements management, system design, development or deployment, and tools used in the operations process for change and issue management.

2.17 Building Block 17: Quality-driven characteristics

Using the four basic elements, leading to Confidence, the fifth element, will create an approach that focuses on product quality: sometimes called 'fit for purpose' and sometimes 'fitness for use'. That is why this approach is called 'quality-driven'. It can be integrated in all kinds of development or project methods, frameworks or approaches. Even better: it will only be successful when integrated, since Integrate is a key element and it cannot work as a stand-alone process.

The quality-driven approach, based on the elements, has certain characteristics that are more or less fulfilled by applying the approach, depending on the approach with which it integrates, because the elements may be applied somewhat differently. Some characteristics are directly related to an element, some follow from combining elements. Many characteristics are somehow linked. The characteristics marked with an arrow (>) are essential to create Confidence.

The characteristics concerning the testing are especially mentioned.

The order of the list does not necessarily mean any ranking in importance, nor is the list necessarily complete.

Characteristics of a quality-driven approach:

> Only features that meet the predefined quality standard are released.

- Direct involvement of users and their management (business driven).
- Test in all stages, start as early as possible.
- Tests are automated where possible and useful, in order to test better, more, and more often.
- Testing at the end is only to demonstrate value, a working solution.
- The role of test professionals evolves: integrated with other disciplines and helping them in all phases, stages and activities, using their test expertise.

> Quality is everybody's concern.

- Tests are used to find faults.

> Quality is built into the process.

> Continuous improvement of the process is built in.

- The people involved are mandated to decide about their own work process to improve quality.
- Every deviation, defect, imperfection is a trigger to improve.
- Open culture where people can trust each other.

> Mindset: an attitude to honor and live all of the above aspects.

- A quality coordinator has a mandate to intervene on quality issues and constantly pay attention to quality, using tests to monitor and check.
- Support from highest level of management.

As an example on how it can work out in practice, the brief action plan of Seabiscuit is shown, as made by Neil, Hal and Francine and used in the story:

Instruments, measures and actions to achieve this:	**Action by**
- Visit to a factory to inspire.	H (M)

-	Overall project management.	H
-	Hire team leaders, experienced in a quality-driven method to coach and build team.	H
-	Agile method (short-cycled, iterative, using demo, retro and definition-of-done).	H
-	Team retrospectives to improve the process for every deviation from quality standard	H
-	Training and teambuilding.	H
-	Coaching of Hal by Mr. Mikkel on the quality-driven approach.	M
-	Select people carefully to build teams.	H (M)
-	Involving Ann as user representative, participating on a daily basis.	N
-	Cross-team retros by Neil (if necessary attending, stimulating team retrospectives).	N
-	Constant attention to quality aspects, (pioneering, communicating, stimulating)	N
-	Monitoring on quality aspects (dashboard).	N
-	Adequate set of quality criteria, used as the standard for defined quality level.	N
-	Use tests to monitor, check the actual state of quality.	N
-	Use of test tools to test more, better and more often.	N
-	Pick up and secure cross-project issues, improvements, experiences (PDCA), and build a long-term policy for ZBO (test expertise, tools, quality-driven approach, etc.)	N
-	Position of Neil independent of the project, acting on behalf of Owen, supporting Ann and Hal, overseeing. Mandated by Rupert to intervene on quality issues (assignment).	N

Responsibility for each point is indicated by initial: H = Hal, N = Neil, M = Mr. Mikkel.

Guarantees:

- High quality is guaranteed by the establishment of quality criteria, which must be met by product features before release.
- A solution with working features can be released every cycle due to the timeboxing structure.
- Quality will rise and costs will decrease over time as a result of continuous improvements.
- The most important features will work at a final deadline after multiple cycles because features are reprioritized every timebox.

2.18 Building Block 18: Integrated Test Organization

How do you organize testing? Many people have struggled. Many different solutions have been found. However there was one basic division of responsibilities. Traditionally we see the 'project organization', 'line organization' and 'staff organization'.

Briefly summarized, the project organization focuses on achieving well-defined one-off goals, the line organization focuses on long-term goals such as maintenance (terms such as 'test factory' or 'test line' are used in this context), the staff organization basically supports people in the project and line organization with specialist expertise such as test tooling (this is sometimes known as a Test Expertise Center).

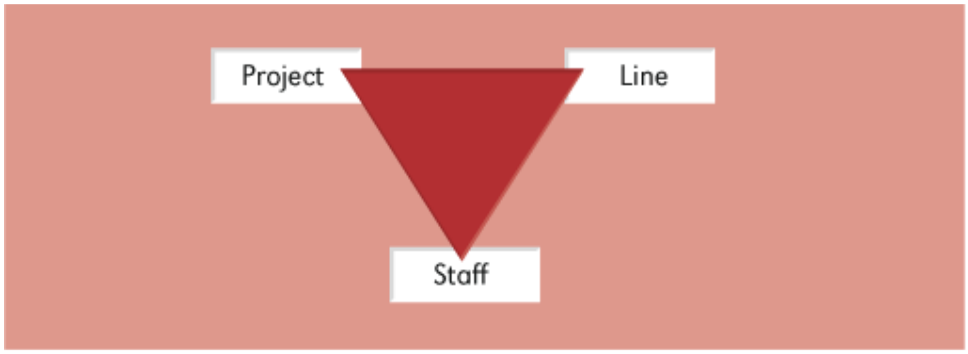


Figure 11. The traditional division of responsibilities in a test organization

In traditional IT, we could see unit testing being organized within projects. System testing would be done by independent testing teams, as a line organization (including an extensive regression test) and the acceptance testing was done by the business representatives that were supported by the testing staff organization.

Nowadays the trend is towards integrating all activities. What does this mean for the test organization?

The modern approach to solving information-technology challenges is to have small self-contained and empowered teams. The ultimate form, known as the 'whole-team approach' or 'DevOps', integrates all design, development, maintenance and operations tasks. So the distinction between project organization and line organization no longer exists. They have blended together.

The benefits of this integrated organization are enhanced communication and collaboration within the team, an elevation of the various skill sets within the team to the benefit of the project, making quality a shared responsibility.

This will work very well in small organizations with only a few teams: all expertise that can effectively support the business needs is available within the teams. But what does this mean for larger organizations?

If your organization has a large number of integrated teams you will come across two challenges:

1. How do the teams effectively exchange necessary information?
2. How do the teams get skills and expertise they don't have within their team?

Re 1) Small empowered teams may tend to isolate themselves, as that helps avoid distraction so that they can keep up their velocity. However, some information must be exchanged to maintain alignment in the results of the teams. In addition, the long-term maintainability of the information systems in a larger organization will benefit from using standards that need to be agreed amongst the teams.

Re 2) In a small empowered team there will typically be one or two people who are particularly skilled in a specific area, such as one business analyst, one systems designer, two programmers, an operations person and a tester, for example. On the subject of testing, the team members will have general knowledge and skills, and the tester in the team will have more in-depth knowledge and skills. However, one single person can't be an expert at all areas of the testing profession, so how does the team get the missing knowledge and skills?

In a small organizations (let's say with 3 teams) everybody still knows each other and, on an informal basis, they will be able to manage the challenges described above.

But in larger organizations both challenges call for support. The 'staff organization' is needed to properly organize this support. The staff organization consists of experts in various fields who are able to support multiple teams in the organization. For example, the staff organization will be called upon to support the teams when setting up an automated regression test, or to do the overall maintenance and support of test management tools. Also the staff organization will create the guidelines and standards that the teams ask for (bear in mind that the staff organization should not just create standards for the sake of standardization, they only do this on request of the teams to solve impediments).

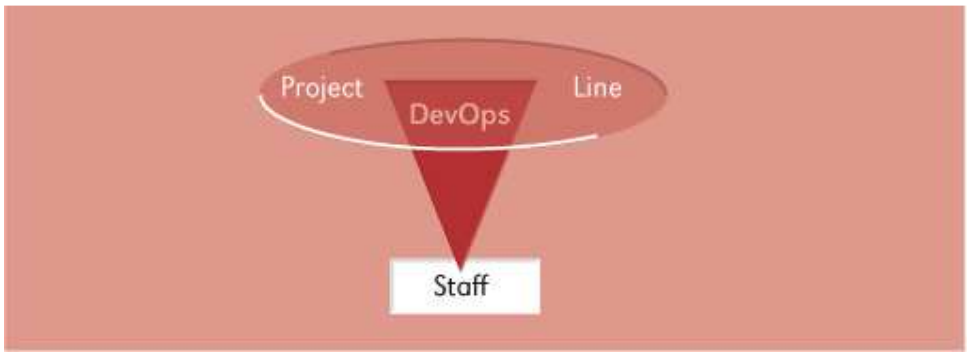


Figure 12. Testing in integrated and empowered teams, supported by the staff organization.

Summarizing, the main reason for having a staff organization is that specific specialist knowledge and skills are too scarce within the teams, so they will have to be added from outside the teams. Thus you will get the optimal benefits of integrated and empowered teams.

Integrate is one of the elements introduced in this book. The integrated organization is the answer to today's challenges. In this way, the assurance of adequate quality is embedded in the activities of the teams. And whenever the team itself lacks certain skills or expertise, the staff organization is available to assist.

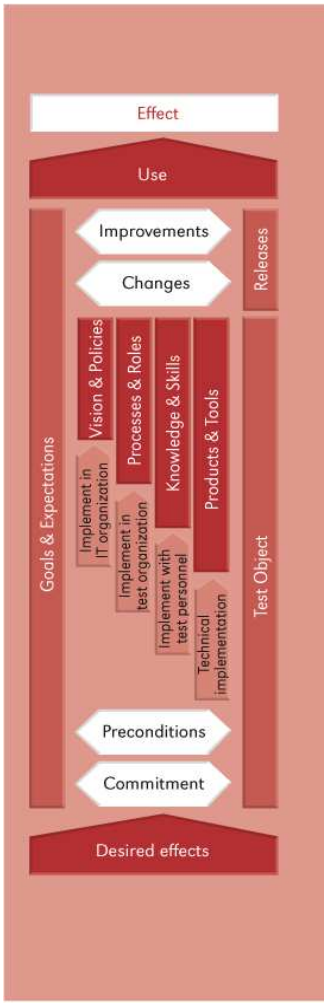
2.19 Building Block 19: Implementing test tools

To achieve the desired effects of using a test tool, we need to implement it. After implementation, the primary effect can be reached straightaway, the derived effects take longer.

How to implement a test tool varies per type of test tool, but there are generic aspects that the implementation of any type of test tool should address.

There is more to it than just installing the tool, as is visualized in the test tool implementation model (see figure 13).

There is always a relationship between the test tool and the test object. Most types of test tools have a strong relationship with the technology of the test object. This technology determines if a test tool can be used and if so: the amount of effort required to implement and maintain a usable solution. Another factor is the number of releases of the test object: this determines the frequency of use of the test tool but also the frequency of having to maintain it.



Setting goals in terms of scope, results and timeframes is an obvious best practice, but dealing with expectations is equally important: they can differ substantially from what will actually be achieved.

The result is that the implementation of a test tool is wedged between the Goals & Expectations and the Test Object & Releases.

The implementation starts with getting commitment and dealing with preconditions. The technical implementation deals with installing and setting up the test tool to create a usable and maintainable solution. A good technical implementation is not enough, people remain the critical success factor, and equipping personnel with the right knowledge and skills to use and maintain the test tool is essential. The implementation is truly successful when using the test tool (and, of course, the necessary activities to be able to keep using it) has become an integral part of the testing process (or even better: the development process). In other words, when using the test tool has become self-evident.

After implementation we can use the test tool, but we have to keep adapting to changes to keep achieving the intended effects. The most obvious changes are changes in the test object that require maintenance in the test tool, but changes in organization or processes need adjustment too. Continuously looking for improvements on all levels helps us increase the effect of using test tools.

Figure 13. Test tool Implementation Model.

Most test tool implementations are initiated on the operational level, giving a bottom-up approach that focuses on the use of a single type of test tool for testing a single application. This is an effective approach, mostly because commitment is more easily achieved and goals are more easily met due to of the operation level of the goals. But embedding the use of test tools in the application lifecycle (the Vision & Policies layer) is more difficult and is often forgotten.

A top-down approach starts with strategic choices that reflect the goals of the entire organization, deals with all types of test tools and integrations, and governs the individual implementations. Although top-down implementation takes longer because of the larger scope, it maximizes the effects of using test tools by integrating them into the entire application lifecycle.

2.20 Building Block 20: Reviewing requirements

The goal of software development is well-functioning (qualitative) end-products and services. Along the way, whether it be waterfall or Agile development, many work (or interim) products are created to get to that goal: business case, requirements, plans, designs, etc. If those work products don't have the right quality, they will never lead to the desired outcome.

Assessing those work products can reveal potentially expensive defects at an early stage. The sooner a defect is found, the simpler and cheaper it can be reworked. The goal should be to detect defects at the source (see the "Root cause analysis and Metrics" building block). Besides cost and lead-time reduction, another advantage is that the gap between the expected and realized result narrows. Assessing work products is referred to by many names such as evaluation, 'reviewing, 'examination', and 'inspection'.

Not only are defects found earlier, some defects are also found more easily than in actual testing: defects like deviations from standards, unclear and inconsistent defects, insufficient maintainability etc.

Work products can be compared with:

- The preceding work product
- Criteria from the succeeding phase (established in checklists)
- Other work products at the same level
- Agreed products standards
- The expectations of the client.

There are various review techniques, varying in purpose, formality responsibilities and procedure. Since not every work product needs to be assessed with the same effort, different techniques can be chosen per work product. For more information, see TMap NEXT, 2006.

Reviewing is not a difficult process to set up, but in practice the process can become mired in practical execution problems. Here is a (non-exhaustive) list of thing to be done or avoided to overcome this:

- Go prepared into review meetings
- Have structured review meetings
- Don't have more than 6 people at a meeting

- Don't have people who are too dominant (functions / roles) at a meeting
- Criticize the product, not the creator
- Register defects properly and analyze root causes (and metrics)
- Take care of support
- Divide large documents to prevent only the first 20 pages being properly assessed
- Vary in assessors to overcome relaxation of attention.

In the course of time the intensity of reviewing may decline, for lessons learned ought to have resulted in fewer major defects.

In more detail

The importance of good requirements to a software project cannot be understated. According to analysts, as many as 71% of software projects that fail do so because of the quality of the business requirements. Requirements mark the boundary between what we would like to have and what we are going to build. Essentially they state the need of the business owner of the project. If there are errors or ambiguities there, then the whole project is at risk.

There are many shapes and forms of requirements, but the important thing is that everyone in a project that handles requirements should ensure that he or she understands what is meant by them. There are many ways to make sure of this. At very least, everyone involved should ask himself the following questions:

- Consistency: Are there requirements that contradict each other in some way?
- Completeness: Do the requirements describe all the attributes of the system to be implemented?
- Verifiability: Is it possible to check if a requirement has been built correctly?
- Traceability: Is it possible to check the status of this requirement in all the stages of software development?
- Atomic: Is this requirement as simple as possible (but not simpler)? An easy check for this is if the requirement contains words like 'and', 'or' and but'.
- Structure: Do all the requirements have the same structure? There are many ways to capture requirement (think of user stories, requirements documents, etc.) but the most important thing is to be as consistent as possible in the way requirements are captured.
- Feasibility: Can the requirement be achieved by the organization given its current state of time, budget and capabilities?
- Understandability: Can everyone involved in the project understand what is meant by the requirement?

There are many more checks that one can do to monitor the quality of the requirements. For instance, it is often helpful to do a check for "weak words". These are words that, when they occur within a requirement, are often part of an ambiguity in the requirements, or signal that a requirement might be unclear. A list of 'weak words' can be found on tmap.net.

Chapter 3 Website

3.1 Introduction

Chapter 3 of this workbook will focus on test design. The source that appears in this section is information from the TMap Suite website (www.tmap.net) on this topic.

Due to the changing nature of the website we have included the information for certification in the workbook. This way the workbook remains the source for the certification, but all topics that are discussed in the workbook, are also discussed online.

3.1.1 Reading Guide

In this chapter a number of topics are covered. In the first two sections the importance of test design is discussed and some key concepts around test design are explained. In sections 3.3 up to and including 3.6 a number of groups of different coverage types are illustrated.

In the final section 3.7, 8 test design techniques are elaborated on.

3.1.2 Why test design?

One of the most important goals of testing is a clear advise on quality and risk in such a way that all the parties involved gain confidence in the product. To be able to do this, a tester has to gather information on system behavior. One of the main tools in gathering information is the executing of test cases. The results of those cases give information on the system behavior. The main questions are: Which test cases? How many? And how do we get those cases? In answering those questions test design is indispensible.

Designing the right set of test cases is the essential link between the test strategy and the implementation of the test strategy - the tests that are executed.

This takes place in the context 'of test assignment to test cases' (also see Workbook: section 4.3 "Planning Phase" or on tmap.net: Test design or section 6.2 "Planning Phase" of TMap NEXT®.)

See figure 14 for the link between the relevant terms in test design:

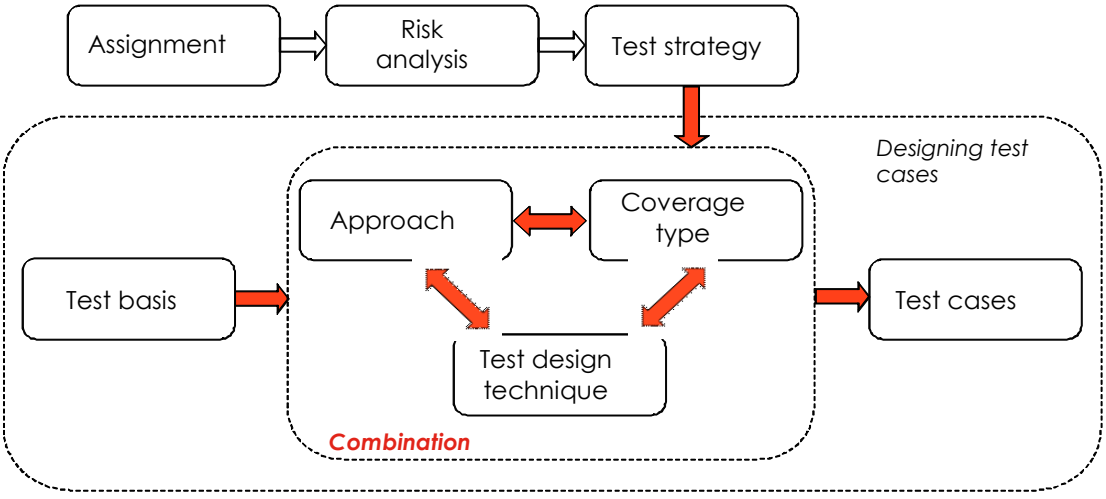


Figure 14. From assignment to test cases.

This is all about making *choices*. The outline of that is (see figure 14):

- It is never possible to test '*everything*'. For example, because of the constraints in terms of time and costs that have been given in the assignment formulation. But also because what that 'everything' now really means can rarely be defined unambiguously (for example: all the lines of code, or any combination of data, or any quality characteristic or all possible paths in the process or any faults or ... or ...?). There are all sorts of choices that must therefore be made.
- The more important an item is, the more thoroughly it must be tested. The importance (choice) of items, such as system parts, is determined by performing a *risk analysis* (for instance PRA of PRBA).
- In the *test strategy* an overview is made of the whole test and how the testing effort is divided between different test varieties (choices) to cover the mentioned risks most adequately. The characteristics of the test object that are under test and the thoroughness with which these must be tested together determine the *coverage* over the test.
- The test strategy has to be translated to test cases to substantiate the test strategy. In many cases, the substantiation of the test strategy has to be *demonstrated* (this can, for example, be one of the preconditions in the assignment).
- How can we design the test case (choice)? This depends on a couple of factors: :
    - The agreed coverage (the characteristics that must be tested and the thoroughness with which these must be tested)
    - The available test basis - information on system behavior on which the test cases are based
    - The way the software development process is organized (for instance waterfall vs. agile)
    - The knowledge and experience of the people involved
    - The time and budget available to execute the tests

    Based on these factors, a choices are made – not in a prescribed order – with regards to test approach(es), in coverage types and test design techniques.

The selection order of Approach(es) Coverage Type(s) and Test Design Technique(s) is not fixed in advance.

For example:
In an Agile environment the Experience-based approach could be chosen first and within that approach Exploratory Testing. In carrying out ET the most appropriate coverage types (Coverage-based approach) are then applied.

In another situation, for example, the Process Cycle Test is immediately chosen on the basis of the required coverage and the available test basis, within which the test depth level is then determined. With this the approach (coverage-based) has automatically been determined.

In yet another situation a number of coverage types is first selected, wherein the thus obtained test situations are combined to test cases. The specific name of the test design technique is no longer relevant (perhaps an entirely new TOT). In this case the approach is also set automatically: coverage based.

Tip: Preferably use a mix of coverage and experience-based approaches.

- This will lead to a set of test cases that will fulfill the test strategy in a way that is needed to complete the assignment.

3.1.3 The Benefits of Test Design According to the TMap Suite

Thorough test design is important. In addition to the above, a number of arguments for this can be mentioned:

- Because test design, at least the coverage-based approach (see Building Block Test Approaches), is aimed at reaching a certain coverage for finding certain types of faults (e.g. interfaces, process, input checks or the processing), such faults will be detected in an effective manner.
- The test design, in most cases, aims to achieve the required coverage with the least possible test cases.
- The tests are reproducible, because the order and content of the test execution have been described in detail.
- The standardized approach makes the test process independent of the person who specifies the test cases and executes them.
- The standardized method makes the test specifications transferable and maintainable.
- The testing process is easier to plan and manage, as test design and execution can be divided into well-defined blocks.

3.2 Framework and Importance of Testing

3.2.1 Introduction

In test design it is all about realizing a set of test cases that demonstrates in the agreed extent the agreed coverage.

Because of this we firstly discuss what a test case actually is.

3.2.1.1 What is a test case?

A test case is used to examine whether the system displays the desired behaviour under specific circumstances. It must therefore contain all of the ingredients to cause that system behaviour and determine whether or not it is correct. A well-known way to describe system behaviour is 'Input → Processing → Output'.

A test case consists of a description of the starting point (also known as initial situation), the test action and the predicted result::

- Starting point (initial situation)
  This covers everything that is needed to prepare the system for receiving the required input. This includes not only the data that are needed for the processing, but also the condition in which the system and its environment must be. For instance, one might think of setting a specific system date, or running specific week and month batches that bring the system to a specific status.
- Actions
  This means all of the activities that must be executed to activate the system to the processing. It might be a simple command ('Run …') or entering specific data on a

screen. But it can also be a complex sequence of entering parameters, activating a specific function, manipulating other data, starting up another function, etc..

- Predicted result
  This covers all of the results that the tester must check to establish whether the system behaviour conforms to the expectations. Often, predicted result is incorrectly thought to be limited to the output that appears on screen or is stored in databases. But the system can also produce output that is transmitted to other systems or peripheral equipment. Furthermore, more than just output data may have to be checked to establish that the system is working correctly. For instance: 'How quickly should the output appear?', 'What is the maximum allowed memory load and is it released afterwards?', or 'Should the system produce interim signals or messages, such as the hourglass or beeps?'

See the figure 15 for the generic structure of a test case, in relation to the system's behaviour under test.

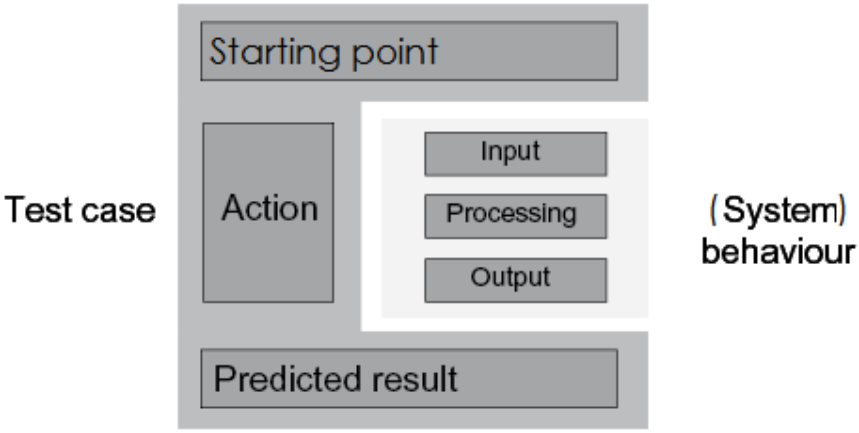


Figure 15. Generic structure of a test case, in relation to the system's behaviour under test.

In other words, executing a test case roughly goes through the following steps: 'Prepare this → Do this → Check that.'

Contrary to a test situation – which addresses an isolated aspect – a test case is a complete unit that can be executed as a separate test.

3.2.1.2 Key Concepts in Test Design

The key concepts in test design are:

- Coverage
  - Coverage Types
  - Thoroughness
- Test Approach
- Test Design Technique

These concepts and their interrelationships are explained in sections 3.2.3 to 3.2.5. In section 3.2.2 is first explained in detail of which generic steps test design consists and how coverage types and test design techniques are related.

3.2.2 The Generic Test Design Steps

The creation of test cases follows the following five generic steps:

1. Identifying test situations
2. Creating logical test cases
3. Creating physical test cases
4. Establishing the starting point
5. Creating test script.

These five generic steps are independent of the chosen test design technique and are *always* applicable. In some cases (because of the chosen test design technique or other circumstances) steps can be skipped or merged.

The relationship between the concepts is shown in figure 16:

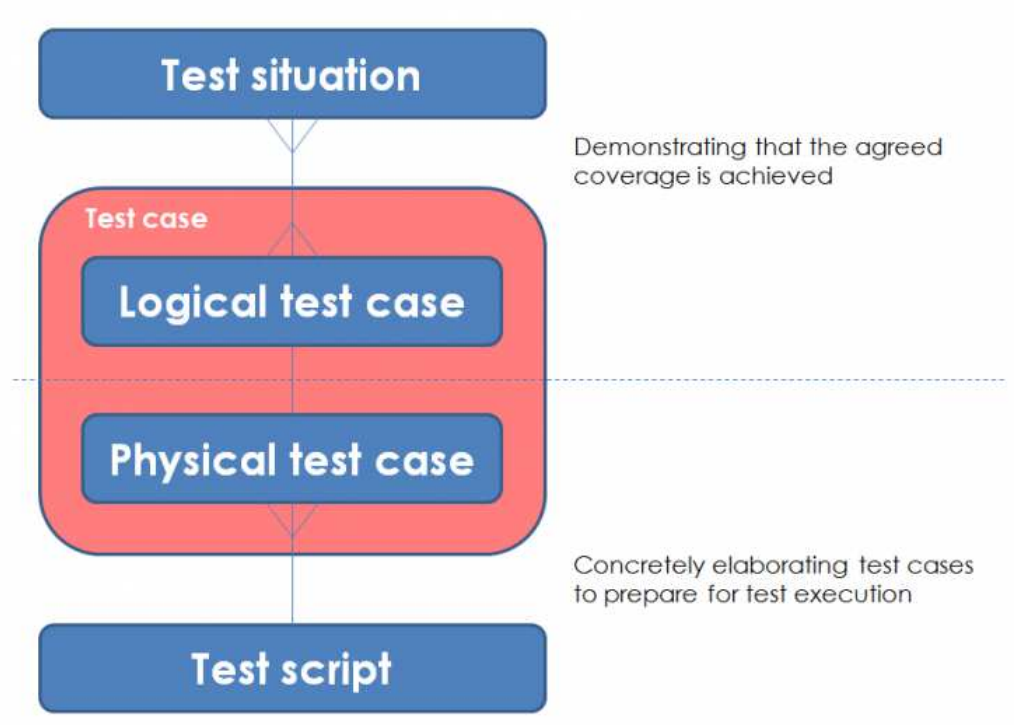


Figure 16. Relations between the concepts test situations – test cases – test scripts.

The relationship between the concepts is shown in the figure above and can be summarized as follows:

- Every test situation occurs in at least 1 test case
- A logical test case covers 1 or more test situations
- Every logical test case is worked out concretely into exactly 1 physical test case
- Every physical test case occurs in 1 test script.

The figure also shows the distinction between the logical and physical parts of the test design:

- The *logical test design* consists of the test situations and the logical test cases. This is the part that demonstrates that the required coverage is achieved, thereby complying with the test strategy.
- The *physical test design* consists of the concretely created physical test cases, laid down in test scripts. This guarantees a thorough preparation of the 'execution' of test cases. The physical creation of test cases therefore adds nothing to the thoroughness of the test.

The Concepts Explained

- **Step 1** of test design is identifying test situations.

  A **test situation** is:

An isolated occurrence (possibility) that must be tested.

In a *coverage-based* approach the test situations, by definition, are obtained by applying one or more **coverage types**. In an experience-based approach testing situations are based on skill, intuition and experience of the tester.

- In steps 2 and 3 the test cases are determined.

  With a **test case**:

*it is examined whether the system displays the desired behavior **under specific circumstances** (the test situations).*

  A test case is from "beginning" (input) to 'end' (output) and contains one or more test situations.

- **Step 2** entails that the test situations are combined to form logical test cases, so that each test situation is covered by at least one logical test case.

  A **logical test case**:

*describes in **logical terms** the circumstances in which the system behavior is investigated, by indicating which **test situations** are covered by the test case*

  In other words, what will be tested, indicated in abstract terms.

- Step 3 involves the logical test cases being sufficiently developed specifically to actually perform the test cases. Choices are made regarding physical values.

  A **physical test case**:

*is the **concrete elaboration** of a logical test case, in which choices are made for the values of all the input and also for the environmental settings*

  Physical test cases usually contain a concrete description of:

  - Initial situation
    - All that is required to be able to receive input from the system, such as:
      - Database with necessary data
      - Environment parameters, e.g. system date
      - State of the system
  - Action
    - All activities required to activate system behavior :
      - Straightforward: run batch program or entering data
      - Complex: a great many actions
  - Result Prediction
    - All results that need to be checked such as:

- Correct message on screen
- Data base changed yes/no

- **Step 4** includes determining the starting point, that is all that is needed to execute the test cases. The starting point for test design includes the initial situations of the individual test cases from the test design, supplemented with everything else that is needed to be able to carry out the set of test cases. The starting point is prepared before the test execution.

  One step further is that the starting points for different tests may also show (big) overlap. For that reason, it often involves one or more central starting points which are for multiple tests of the application.

- **Step 5** is the preparation of the test script. In this document the test actions and checks of the physical test cases have been described in the most optimal test execution order. These test cases must not be able to disturb each other. The test script as such is the roadmap for the test execution and also offers the possibility for monitoring progress. The physical test cases and starting point naturally form the basis for manufacturing the test script.

  The general contents of a script is as follows:

  - Unique identifier, consisting of:
    - version;
    - author;
    - test basis including version.
  - Preparing the starting point
    For example by setting the system date, restoring a certain back-up and adding certain test data
  - Test actions and –checks
    The physical test cases in a suitable sequence for execution, with for each test case, the required initial situation, action and outcome monitoring. When a good starting point is set up, nothing needs to be done anymore for the initial situation.
  - Restoring environment
    Ensure that the results of the executed test, if necessary, are restored again so that other testers experience no disruption (think also for example, of restoring the system date).

3.2.3 Coverage, coverage types and test intensity

3.2.3.1 Coverage

The choices in your test strategy on WHAT to test indicate that you want to cover certain aspects of the test object. The objective of an effective test strategy is therefore to realize the best achievable coverage at the right place. Coverage has everything to do with the wish to find the most possible defects with the fewest possible test cases.

But what is coverage? Coverage is very subjective. We cannot talk about the coverage. What does an executive or other stakeholder mean when he/she asks you what the coverage of the test was? What information does he/she need or want? Possibly he/she wants to know how thorough some aspects of the test object have been covered. Maybe he/she want to know how many of all possible defects have actually been found by the tests.

A key word here is Coverage. A definition for coverage is hard to give. It basically deals with aspects of the test object that you would like to assess and the thoroughness with which you do that.

More important is the question if we are able to achieve 100% coverage. Well, we can never be certain that all defects have been found or even that 60% of all defects has been found. After all, we do not know how many defects there actually are. Furthermore we don't know how accurate and complete the information was on which we based our test cases. Also if the tests we executed were based on a test strategy (and product risk analysis) we can never be sure whether our stakeholders made the right choices on what to cover. Testing everything is simply impossible, because it is impossible to define what 'everything' means.

Although coverage is hard to define, it has a relation with the following two terms:

- The aspects of the test objects (e.g. quality characteristics) that must be tested

and

- Coverage thoroughness applied to each of those parts.

3.2.3.2 Coverage types

The definition of a coverage type is:

the form in which test situations are deducible from the test basis.

This concerns:

- the options that need to be tested
- *and* the method of working to identify those options.

A coverage type focuses on achieving a specific coverage to detect specific types of defect (e.g. in the interfaces, the input checks or the processing), such defects are detected more effectively then by specifying ad hoc test cases. One coverage type could only be called 'better' with any practical use if it would find at least all of the defects found by the other coverage type plus some additional defects.

Summarized (see figure 17):

- It is not possible to test everything within the confines of the preconditions of time and costs defined in the assignment. Choices will have to be made as to the lengths one wishes to go to in testing.
- A test strategy is used to create an overview of what will be tested and how thorough, such that the aspects to be tested are covered as adequately as possible.
- The decisions concerning thorough and less thorough testing are translated to concrete statements about the targeted coverage.
- Depending on the available test basis, among other things, appropriate coverage types are selected to achieve said coverage

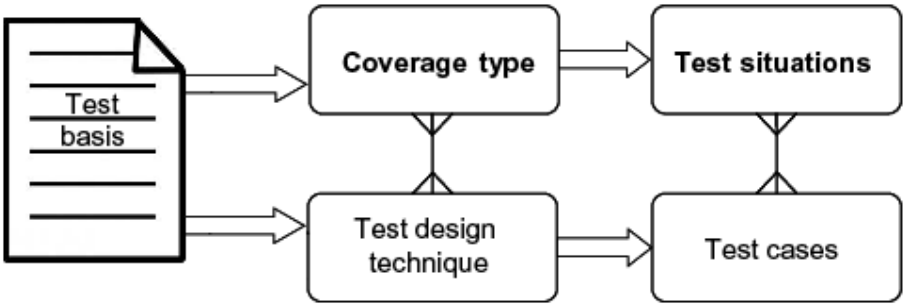


Figure 17. Summary of deriving test cases.

3.2.3.3 Coverage thoroughness

In the test strategy is decided what test intensity is to be achieved in the test. Along with the aspects of the test object to be assessed this indicates what kind of coverage is to be achieved and with what thoroughness.

This means that executives and other stakeholder most likely expect information about the thoroughness of your test. But what do they mean? There is no unambiguous definition for test intensity. It is about aspects such as:

- How thorough was the chosen coverage type?
- Were multiple coverage types applied?
- How high was the variation of thoroughness within a specific coverage type?

It is not a question of 'better'!

So, although a fascinating subject, it's also a complex matter. There is no black and white here. However, we can state that testing everything is impossible. How more thorough is a certain coverage type compared to another (e.g. pairwise testing versus modified condition/decision testing)? How more thorough is one variation within a coverage type compared with another variation within that same coverage type (e.g. modified condition/decision testing versus modified condition coverage)? How many additional defects are to be found?

3.2.3.4 Coverage groups

Coverage types can be divided into four *coverage groups*:

> **Process**: Processes can be identified at several levels. There are algorithms of control flows, business processes. Coverage types like paths, statement coverage, and state transitions can be used to test (variations in) these processes.
>
> **Conditions:** With almost every system, there are decision points consisting of conditions, where the system behaviour can go in different directions, depending on the outcome of such a decision point. Variations of these conditions and their outcomes can be tested using coverage types like decision coverage, modified condition/ decision coverage, and multiple condition coverage.
>
> **Data:** Data is created and ends when it is removed. In between, the data is used by updating it or consulting it. This lifecycle of data can be tested, but also combinations of input data, as well as the attributes of input or output data. Some coverage types here are Boundary values, CRUD, Data flows, and Syntax.

Appearance: How a system operates, how it performs, what it's appearance should be, is often described in non-functional requirements. Within this group we find coverage types like operational and load profiles, and presentation.

3.2.3.5 Coverage types per coverage group

The table below gives a brief description of each coverage type per group.

In the next section an indication is given how the coverage thoroughness can be varied within several coverage types.

GROUP	COVERAGE TYPE	DESCRIPTION
Process	Control flow	Testing the program structure.
	Paths	Coverage of the variations in the process in terms of combinations of paths. A scheme of decision points and paths is required as a test basis.
	Rare events	Addressing events that happen very infrequently
	Right paths/ fault paths	Checking both the valid and invalid situations in every defined error situation. An invalid situation (faulty control steps in the process or algorithm that precede the processing) should lead to correct error handling, while a valid situation should be accepted by the system without error handling.
	State transitions	Verification of relationships between events, actions, activities, states and state transitions.
Conditions	Decision points	Coverage of the various possibilities within a decision point with the purpose of arriving at the outcomes of TRUE and FALSE
	Semantics	Validation relationships between data.
Data	Boundary values	A boundary value determines the transfer from one equivalence class to the other. Boundary value analysis tests the boundary value itself plus the value

GROUP	COVERAGE TYPE	DESCRIPTION
		directly above it and directly below it.
	CRUD	Coverage of all the basic operations (Create, Read, Update, Delete) on all the entities.
	Data combinations	Testing of combinations of parameter values. The basis are Equivalence classes.
	Data flows	Verifying information of a data flow, which runs from actor to actor, from input to output.
	Domain testing	Coverage of a small number of values from a nearly infinite group of candidate values. Domain knowledge plays a very critical role while testing domain-specific work.
	Equivalence classes	The value range of a parameter is divided into classes in which different system behaviour takes place. The system is tested with at least 1 value from each class.
	Integrity rules	Checking the preconditions under which certain CRUD processes are or are not permitted.
	Right paths/ fault paths	Checking both the valid and invalid situations in every defined error situation. An invalid situation (certain values or combinations of values defined that are not permitted for the relevant functionality) should lead to correct error handling, while a valid situation should be accepted by the system without error handling.

GROUP	COVERAGE TYPE	DESCRIPTION
	Syntax	Validation of attributes of input or output data.
Appearance	Heuristics	Evaluation of (a number of) usability principles.
	Load profiles	Simulation of a realistic loading of the system in terms of volume of users and/or transactions.
	Operational profiles	Simulation of the realistic use of the system, by carrying out a statistically responsible sequence of transactions.
	Presentation	Testing the layout of input (screens) and output (lists, reports).

3.2.3.6 Variations in coverage types

The decision to test more 'thoroughly' can be formalized basically in 3 ways by varying in coverage types:

- a coverage type that is more thorough;
- multiple coverage types;
- a more thorough approach within a specific coverage type.

For some coverage types, it is possible to vary the coverage thoroughness within the coverage type.
The table below gives several examples.

COVERAGE TYPE	VARIATION
Control flow	• Statement coverage • Decision coverage (branch testing/ arc testing) • Paths (see Paths)
Paths	Test depth level N
State transitions	• 0-switch • 1-switch

COVERAGE TYPE	VARIATION
	• 2-switch
Decision points	• Condition coverage • Decision coverage • Condition/ decision coverage • Modified condition/ decision coverage • Multiple condition coverage • Cause Effect Graph • Pairwise testing
Semantics	See decision points and equivalence classes
Boundary values	• Light (boundary value + one value) • Normal (boundary value + two values)
Data	• Right paths/Fault paths • No data pairs • One or some data pairs • N-wise (extension of pairwise) • All possible combinations
Integrity rules	See decision points and CRUD
Syntax	See individual test situations

3.2.4 Test approaches

In addition to what has already been shown in Section 2.9 (Test Approaches) we discuss some more topics in this section.

A test approach is:

> *The test approach is the approach that someone takes when creating test cases.*

There are roughly two approaches to creating test cases:

1. Experience-based
2. Coverage-based

- Preferably use a mix of coverage and experience-based approaches.
- Coverage based approach:
  - Test situations are deduced from the test basis with the aid of *coverage types*
  - Focused on *effective* and *efficient* collection of information about quality and risks
  - Aimed at *provably* achieving the *coverage* that has been *agreed upon* in the test strategy.
- Experience-based approach:
  - Allows the tester to design / think of test cases *prior to* and / or *during* test execution
  - Based on *skills, intuition* and *experience* of the tester
  - Also aimed at the realization of the test strategy, but less certainty about the *actual coverage*
  - Coverage more difficult to *demonstrate*
  - Always a valuable *addition* to coverage-based approach.

3.2.5 Test design techniques

A test design technique is:

> *A standard method to derive test cases from a certain test basis to achieve a certain coverage.*

The importance of the use of test design techniques is represented by the following arguments:

- The tests are reproducible, because the order and content of the test execution have been described in detail.
- A standard way of working creates independence of the test design and the person designing the tests.
- The standard way of working makes sure that the test specification is transferable and maintainable.
- The testing process is easier to plan and manage, because the processes of test specification and execution can be divided into well-defined blocks.

In the ideal situation we would have the certainty, thanks to the test that the system exhibits the correct or desired behavior *under all circumstances*. In reality, not all conditions will be tested, but only a subset that is a direct result of the decisions and choices in the test design.

The generic steps of test design, and thus of the applying of test design techniques are described in section 3.2.2.

3.2.5.1 Relationship between coverage type and test design techniques

E A test design technique is used to derive the necessary test cases that achieve the required coverage from a specific test basis. The first step of a test design technique is the identification of test situations. The test situations are derived by applying **coverage types**. A test design technique suggests the application of one of more coverage types, and subsequently gives directions on how to turn the test situations derived by these coverage types into test cases. Each test situation is covered by at least one test case.

The required coverage is expressed in the selected coverage types. Each coverage type requires a specific type of information in the test basis, e.g. a structured flow chart with paths and decision points..

3.2.6 Selection of coverage types and test design techniques

There exist many coverage types and test design techniques. For the sake of simplicity and practicality we will only highlight the most commonly used test design techniques and hence the application of the underlying coverage types.

To give you a practical overview we highlight the most commonly used coverage types and some test design techniques in which they can be applied.

GROUP	TEST INTENSITY: LIGHT	TEST INTENSITY: AVERAGE	TEST INTENSITY: THOROUGH
Condition	Condition Decision Coverage– Elementary Comparison Test	Modified Condition Decision Coverage – Elementary Comparison Test or Condition decision coverage– Decision Table Test	Multiple Condition Coverage - Elementary Comparison Test or Multiple Condition decision coverage – Decision Table Test
Data	One or some data pairs –Data Combination Test	Pairwise – Data Combination Test	N-wise or all combinations – Data Combination Test
Process	Statement coverage and Paths test depth level 1 –Process Cycle Test	Decision coverage and Paths test depth level 2 – Process Cycle Test	Paths test depth level 2 – Algorithms Test and Paths test depth level 3 – Process Cycle Test

Note: The coverage group "Appearance" is not mentioned here. In cases where this coverage group applies, the coverage type and test intensity are too dependent on the specific situation and the result that the tester wants to achieve.

3.3 *Coverage Types Process*

3.3.1 Introduction

Processes can be identified at different levels. There are algorithms for control flows and business processes. Coverage types in this group can be used to test (variations in) these processes.

This group consists of the following coverage types:

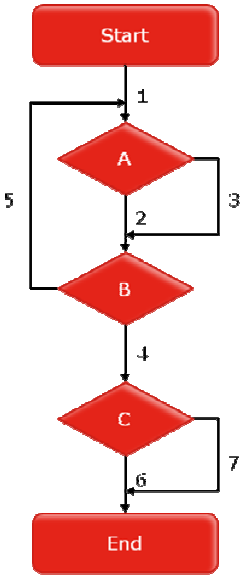
- Paths (section 3.3.2)
- Control Flow (is not explained separately in this Workbook)
- Right paths/Fault paths (is not explained separately in this Workbook)
- State Transitions (is not explained separately in this Workbook)
- Rare Events (is not explained separately in this Workbook)

3.3.2 Paths

Characteristics

Approach	Coverage based - process
Quality characteristic / Test variety	• Functional test • Suitability o for work processes • Code structure • Security
Test Basis	Flow with paths and decision points

Description



The coverage of paths is applicable if the system behaviour is described with the aid of decision points and paths. The figure shows an example of this situation.

Charts of decision points and paths show, in a structured way, how the process runs from start to end and what the various possibilities in the course of the process are: At each decision point, the process can go various ways, indicated by the various paths that continue from the particular decision point. The conditions under which it takes one path or another are described in the decision points themselves.

The aim of the coverage type described here is to cover the variations in the process run that are possible according to the chart. The **test situations** (within coverage type Paths this is also called **path combinations**) are described in this case by indicating which paths in the chart should be followed consecutively.

Keep in mind that such charts with decision points and paths do not necessarily have to be about the functionality of the system. Security processes or work procedures in business processes can also be described with such charts, which makes the basic technique described here applicable to other test types.

The level of abstraction is irrelevant: coverage type paths is applicable to both detailed level (code algorithm) as well as overall system or business process level, as long as the information about the desired system behavior but was given in the structured form of decision points and paths.

3.3.2.1 Light or thorough coverage: the test depth level

In the coverage of paths, various levels are possible. The more thorough the level, the greater the probability of finding defects. This is explained below.

The most elementary form of path coverage only provides the guarantee that each path has been travelled once. The test situations consist in this case of every individual path. Going through the process from "Start" to "End", covering only each individual path, will finds all the faults that will always occur in a particular path. However, it is not for certain that faults that only occur with a *specific combination of process steps* will be found in this way. E.g. a particular fault may be present, which only occurs if path 2 is carried out immediately after path 5.

In order to find this type of fault, testing has to be more thorough. The coverage thoroughness is reflected in the concept of test depth level:

Definition Test depth level N = the certainty that all the combinations of N consecutive paths are covered.

The test depth level in principle runs from 1 to unlimited. The higher the test depth level, the greater the certainty that even faults that occur in complex compositions of process steps will be found. A higher test depth level implies a lower test depth level. In other words, a higher test depth level will at any rate find all the faults that can be found with a lower test depth level, plus possible additional faults.

Deriving the test situations for test depth level 1 is easy: every single path is a test situation. In the example those are the paths 1 up to an including 7.

The basic technique for obtaining test depth level 2 is described below. Subsequently, it is explained how higher test depth levels can be derived simply from test depth level 2.

3.3.2.2 Obtaining test situations with test depth level 2

Irrespective of the test depth level, the starting point for this technique requires a test basis that describes the system behaviour in terms of decision points and paths.

The following steps are then carried out:

1. Decision points & paths
   Nominate the decision points in the process scheme (A, B, etc.) and number the paths. Sum up, per decision point, the:
   a. Incoming paths ("IN")
   b. Outgoing paths ("OUT")
2. Path combinations
   Working out all the combinations of "IN" and "OUT" at each decision point. With a number of incoming P paths and outgoing Q paths, this leads to P times Q path combinations.

When applied to the example:

Decision Point	IN	OUT	Test Situations (path combinations)
A	1, 5	2, 3	1-2; 1-3; 5-2; 5-3
B	2, 3	4, 5	2-4; 2-5; 3-4; 3-5
C	4	6, 7	4-6; 4-7

3.3.2.3 Deriving test situations for higher test depth levels

For higher test depth levels, the following simple mechanism is used:

- Use the list of path combinations of the preceding test depth level as a basis
- Extend each path combination by every possible subsequent step in the course of the process.

It may be formulated as:

Test depth level (N+1) = Test depth level N + "1 step further in the course of the process".

Test depth level 3 can be worked out as follows for our example:

From the scheme the following can be easily deduced:

- Path 1 is the starting point for every test case;
- Paths 2 and 3 are followed by paths 4 and 5;
- Path 5 is followed by paths 2 and 3;
- Path 4 is followed by paths 6 and 7;
- Paths 6 and 7 are end paths and have no successor.

With this information the path combinations of test depth level 2 can be extended to test depth level 3:

	Path combinations of test depth level 2	Extended to test depth level 3
A:	1-2	1-2-4; 1-2-5
	1-3	1-3-4; 1-3-5
	5-2	5-2-4; 5-2-5
	5-3	5-3-4; 5-3-5
B:	2-4	2-4-6; 2-4-7
	2-5	2-5-2; 2-5-3
	3-4	3-4-6; 3-4-7
	3-5	3-5-2; 3-5-3
C:	4-6	No extension and already covered.
	4-7	No extension and already covered.

In the same manner the test situations can be extended from test depth level 3 to test depth level 4.

The ways in which test situations can be combined to realize test cases by coverage type paths are described in section 3.7.3 (Process Cycle Test).

3.4 Coverage Types Conditions

3.4.1 Introduction

In almost every system there are decision points, consisting of conditions, where the system behavior can go in several different directions, depending on the outcome of such a decision point.
Variations of such conditions and the corresponding results can be tested by making use of various cover types, as shown below.

Coverage types within this group are:

- Decision Points (section 3.4.2)
  - Condition coverage
  - Decision coverage
  - Condition/Decision coverage
  - Modified Condition/Decision coverage
  - Multiple Condition coverage
- Semantics (section 3.4.3)

3.4.2 Decision Points

Characteristics

Approach	Coverage based – Conditions
Quality characteristic / Test variety	• Functionality • Security
Test Basis	Functional design
Coverage types	• Condition coverage • Decision coverage • Condition/Decision coverage • Modified Condition/Decision coverage • Multiple Condition coverage

Description

With almost every system, there are decision points, where the system behaviour can go in different directions, depending on the outcome of such a decision point.

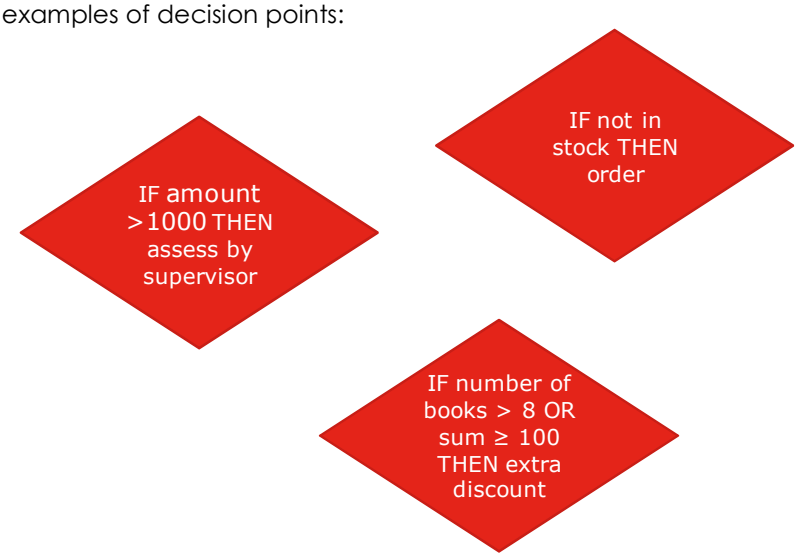
A decision point is:

> *a combination of one or more conditions that define the conditions for the various possibilities in the subsequent system behaviour.*

The various conditions collectively determine the outcome of the decision point. The way in which a condition contributes to the outcome is reflected in terms such as "AND" or "OR". There is a special kind of mathematics – Boolean algebra, or proposition logic – for

the manipulation of these types of constructions. This chapter employs the theory of Boolean algebra, but the intention is not to instruct on this, and the interested reader is referred to the countless books on this subject. Below are the most important basic principles of Boolean algebra that are necessary for the techniques for covering decision points.

Some examples of decision points:



Or take, for example, the following decision point that consists of only one condition:

IF (Number of books > 8) THEN extra discount

Decision points that consist of such **singular conditions** lead to **two** test situations, namely the situation in which the condition is true and the situation in which the condition is false.

In Boolean algebra, 0 is used to indicate that something is false; 1 is used if something is true. In our example, this refers to the following test situations:

Test situation	**1**	**2**
Number of books	> 8	≤ 8
Result	**True (1)**	**False (0)**

Decision points can also consist of combinations of conditions, the so-called **compound condition**. Compare the following compound conditions:

IF (Number of books > 8 OR sum ≥ €100) THEN extra discount

and

IF (Number of books > 8 AND sum ≥ €100) THEN extra discount

Often an abbreviation is used by replacing the conditions by a capital letter (A, B, etc.) The two decision points mentioned above are thus abbreviated to:

A OR B and
A AND B

A compound condition is also either true or false, depending on the truth values of the individual conditions and the way in which the conditions are connected (the so-called **operators**): by an AND or an OR. With two conditions, the following combinations are possible:

A	**B**
1	1
1	0
0	1
0	0

This is called the **complete decision table**.

In the 0-0 situation, both statements are false. In the 0-1 situation and the 1-0 situation, only one of the two statements is true and in the 1-1 situation, both are true. The end result in each of the 4 situations depends on the operator "AND" or "OR": with an "AND" the end result of two conditions is only true if both individual conditions are true; in all the other cases, the end result is false. With an "OR" the reverse is the case: the end result is only false if both individual conditions are false; in all the other cases the end result is true.

The tables below show the outcomes of all situations of a full decision table. Such a table is called a **truth table**. Some examples:

- OR:

A	B	A OR B
1	1	1
1	0	1
0	1	1
0	0	0

  With the operator OR the end result is ONLY ***false*** when ***both*** conditions are false.

- AND:

A	B	A AND B
1	1	1
1	0	0
0	1	0
0	0	0

  With the operator AND the end result is ONLY ***true*** when ***both*** conditions are true.

- Several operators:

A	B	C	(A AND B) OR C
1	1	1	1
1	1	0	1
1	0	1	1
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	0

  In a combined decision there may be different operators. When there are no brackets, AND proceeds OR.

3.4.2.1 Condition coverage

With Condition coverage the possible outcomes of ("true" or "false") for each condition are tested at least once. This means that each individual condition is one time true and false. In other words we cover all conditions, hence condition coverage.

The outcome of the decision point is only relevant for checking the conditions. Also the combinations of conditions are not relevant. Since there are only two possible outcomes of a condition (true or false), condition coverage results in 2 test situations per decision point.

In practice this coverage type is not used very often for the testing of the combinations of the conditions and/or the outcome of the decision point itself is considered to be more important..

Truth table

A	B	A OR B
1	1	1
1	0	1
0	1	1
0	0	0

Condition coverage
The possible outcomes (”true” or “false”) of each condition are tested at least once.

IF number of books > 8 **OR** sum ≥ 100 THEN extra discount

	Number of books >8	Sum ≥ 100	Outcome
TS1	**1**	0	1 (extra discount)
TS2	**0**	1	1 (extra discount)

The condition Number of books > 8 is one time TRUE and one time FALSE

	Number of books >8	Sum ≥ 100	Outcome
TS1	1	**0**	1 (extra discount)
TS2	0	**1**	1 (extra discount)

The condition Sum ≥ 100 is aslo one time TRUE and one time FALSE

	Number of books >8	Sum ≥ 100	Outcome
TS1	1	0	1 (extra discount)
TS2	0	1	1 (extra discount)

Notice that the outcomes of the decision do not need to vary

	Number of books >8	Sum ≥ 100	Outcome
TS1	1	0	1 (extra discount)
TS2	0	1	1 (extra discount)

For Condition coverage only two test situations are needed per decision point

3.4.2.2 Decision coverage

With Decision coverage the possible outcomes of the decision are tested at least once. This means that the result of the decision is one time true and false. In other words we cover one time the THEN and one time the ELSE.

It is relevant to vary in the outcome of the decision, not necessarily in that of the conditions. Since there are only two possible outcomes of a decision (THEN or ELSE), decision coverage results in 2 test situations per decision point.

Truth table

A	B	A OR B
1	1	1
1	0	1
0	1	1
0	0	0

Decision coverage
The possible outcomes (”true” or “false”) of the decision are tested at least once.

IF number of books > 8 **OR** sum ≥ 100 THEN extra discount

	Number of books >8	Sum ≥ 100	Outcome
TS1	0	1	**1** (extra discount)
TS2	0	0	**0**

The outcomes of the decision is one time TRUE and one time FALSE

	Number of books >8	Sum ≥ 100	Outcome
TS1	0	1	1 (extra discount)
TS2	0	0	0

Notice that the outcomes of the individual conditions do not need to vary

	Number of books >8	Sum ≥ 100	Outcome
TS1	0	1	1 (extra discount)
TS2	0	0	0

For Decision coverage only two test situations are needed per decision point

3.4.2.3 Condition / Decision coverage

With Condition/ Decision coverage the possible outcomes of each condition and of the decision are tested at least once. This implies both Condition coverage and Decision coverage. In other words we cover that all conditions are one time TRUE and one time FALSE and we cover one time the THEN and one time the ELSE.

Here it is relevant to vary in the outcome of the decision, and in the outcomes of the conditions. Since there are only two possible outcomes of a decision (THEN or ELSE), and there are only two outcomes of a condition, test situations can be created in such a way that only 2 test situations per decision point are needed.

Truth table

A	B	A OR B
1	1	1
1	0	1
0	1	1
0	0	0

Condition/Decision coverage
The possible outcomes ("true" or "false") of each condition and of the decision are tested at least once.

IF number of books > 8 **OR** sum ≥ 100 THEN extra discount

	Number of books >8	Sum ≥ 100	Outcome
TS1	**1**	1	1 (extra discount)
TS2	**0**	0	0

The condition Number of books > 8 is one time TRUE and one time FALSE

	Number of books >8	Sum ≥ 100	Outcome
TS1	1	**1**	1 (extra discount)
TS2	0	**0**	0

The condition Sum ≥ 100 is aslo one time TRUE and one time FALSE

	Number of books >8	Sum ≥ 100	Outcome
TS1	1	1	**1** (extra discount)
TS2	0	0	**0**

The outcomes of the decision is one time TRUE and one time FALSE

	Number of books >8	Sum ≥ 100	Outcome
TS1	1	1	1 (extra discount)
TS2	0	0	0

For Condition/Decision coverage only two test situations are needed per decision point

3.4.2.4 Modified Condition / Decision coverage

With Modified Condition/ Decision coverage (MCDC) every possible outcome of a condition determines the outcome of the decision *at least once*. In other words we cover that each condition when TRUE determines a TRUE outcome of the whole decision point, and when FALSE determines a FALSE outcome of the whole decision point. This implies Condition Decision Coverage.

MCDC guarantees:

- That there is at least 1 test situation in which the outcome is TRUE, owing to the fact that condition A is TRUE
- That there is at least 1 test situation in which the outcome is FALSE, owing to the fact that condition A is FALSE
- The same goes for all other conditions in the decision point.

This is a thorough level of coverage, with which the following faults, for example, would be detected in the system under test:

- There is a condition missing that should be present
- The "AND" was wrongly implemented as an "OR", and vice versa
- A condition has been inverted, such as "<" instead of ">" or "≠" instead of "=".

The big advantage of this coverage type is its efficiency: with a decision point that consists of N conditions, usually only **N+1** test situations are required for MCDC. Compared with the maximum number of test situations (the complete decision table) of 2N , that is a considerable reduction, particularly if N is large (complex decision points). This combination of "thorough coverage" with "relatively few test situations" makes this coverage type a powerful weapon in the tester's arsenal.

According to the definition of MCDC, every condition should determine the outcome of the decision once. Then all the other conditions in that situation should be given a value that has no influence on the outcome of the decision. This value is called the **neutral value**".

Main characteristics summarized:

- N+1 test situations
- Uses the term "neutral value"
  - Value of a condition (0 or 1) that does not affect the outcome of the decision point
    - should apply for both possible outcomes of the determining condition
    - depends on OR or AND

Determining the neutral value

► Neutral value for AND

A	AND	B	Outcome
1		.	1
0		.	0

Let's say **A** is the determining condition. **A** can be true and false.
So when **A** is true, the outcome has to become true.
When **A** is false, the outcome has to become false.

A	AND	B	Outcome
1		.	1
0		.	0

On these places a value has to be added that has no influence on the outcome of the decision point
(a *neutral* value)

A	AND	B	Outcome
1		1	1
0		~~0~~/1	0

When A is false both the value true and false can be added here. But since the neutral is a value that should apply for both possible outcomes of the determining condition, we chose true.

A	AND	B	Outcome
1		1	1
0		~~0~~/1	0

Neutral value of AND is 1

► Neutral value for OR

A	OR	B	Outcome
1		.	1
0		.	0

Let's say **A** is the determining condition. **A** can be true and false.
So when **A** is true, the outcome has to become true.
When **A** is false, the outcome has to become false.

A	OR	B	Outcome
1		.	1
0		.	0

On these places a value has to be added that has no influence on the outcome of the decision point

A	OR	B	Outcome
1		0/~~1~~	1
0		0	0

When A is true both the value true and false can be added here. But since the neutral is a value that should apply for both possible outcomes of the determining condition, we chose false.

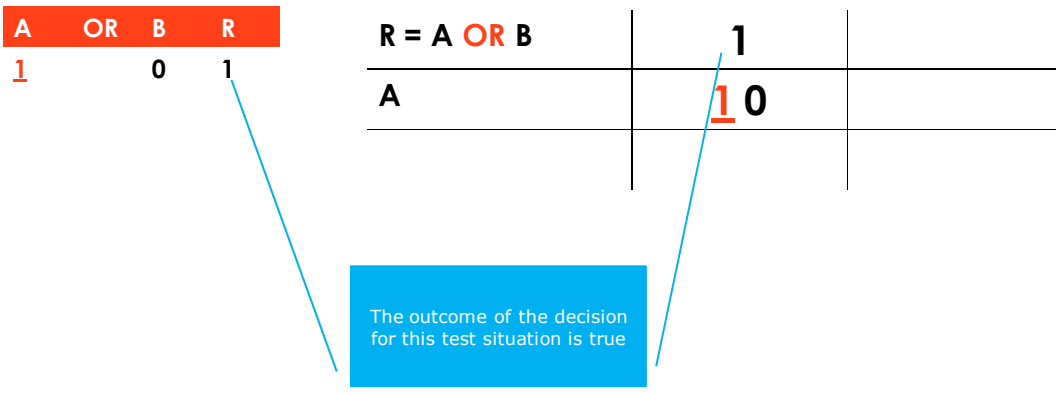
A	OR	B	Outcome
1		0/~~1~~	1
0		0	0

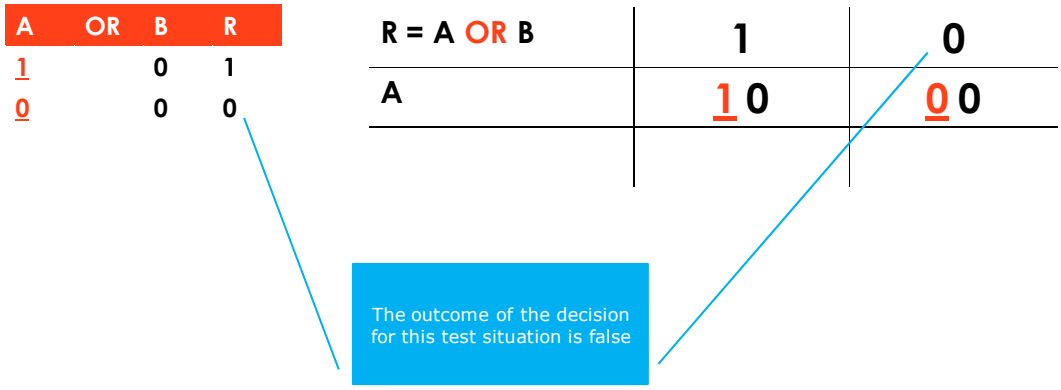
Neutral value of OR is 0

Two ways of notation

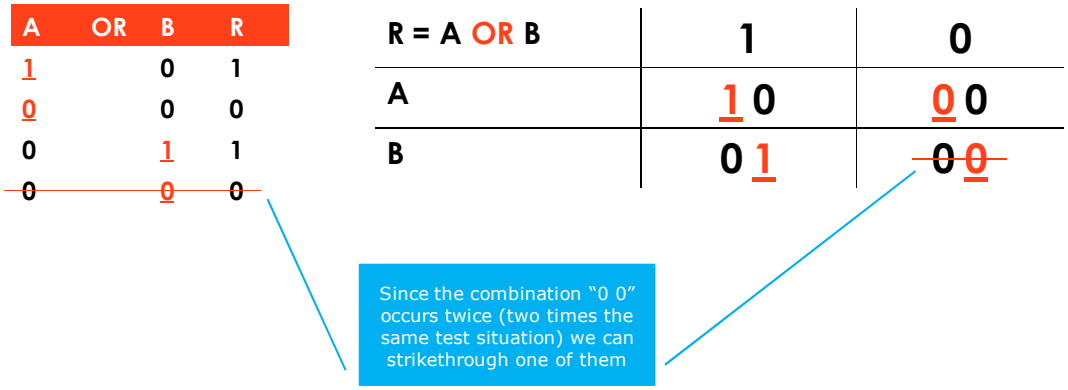
Below, one way of notation is presented on the left side and the other (most commonly used) way is presented on the right side.

- **A as the determining factor**





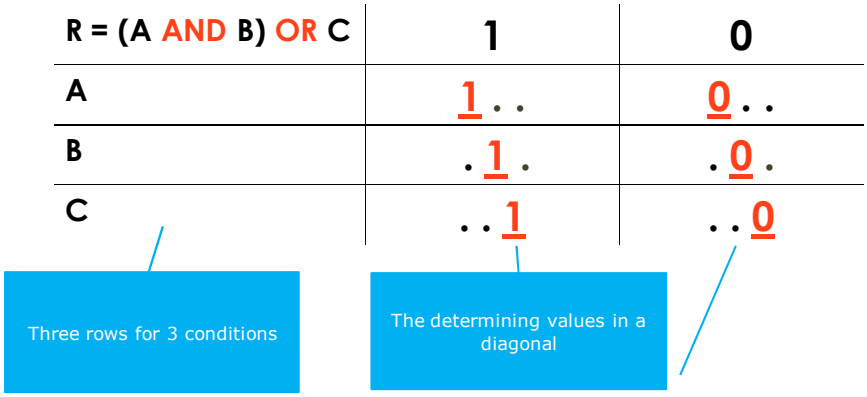
- **Also applied to B as the determining factor**



A more complex example

IF (type of car = delivery van **AND** first use ≥ 1 July 2006) **OR** entrepreneur = no

THEN Tax liable



R = (A AND B) OR C	1	0
A	1 1 .	0 . .
B	. 1 .	. 0 .
C	. . 1	. . 0

The determining value **A** is connected with **B** (between brackets) by the operator **AND**. The neutral value of AND is 1 (true)

R = (A AND B) OR C	1	0
A	1 1 0	0 1 0
B	. 1 .	. 0 .
C	. . 1	. . 0

The combination of **A** AND **B** is true and is connected to **C** by the operator **OR**. The neutral value of OR is 0 (false)

Since they are neutral values we can add the same values here

R = (A AND B) OR C	1	0
A	1 1 0	0 1 0
B	1 1 0	1 0 0
C	. . 1	. . 0

The determining value **C** is connected with the combination of **B** and **B** (between brackets) by the operator **OR**. This combination has to become false (0)

R = (A AND B) OR C	1	0
A	1 1 0	0 1 0
B	1 1 0	1 0 0
C	. 0 1	. . 0

We continue the consistent application of the neutral value by making one of those conditions false. E.g. B

R = (A AND B) OR C	1	0
A	1 1 0	0 1 0
B	1 1 0	1 0 0
C	1 0 1	. . 0

Also here we continue the consistent application of the neutral value. **B** is connected with **B** by the operator **AND**. The neutral value of AND is 1 (true)

R = (A AND B) OR C	1	0
A	1 1 0	0 1 0
B	~~1 1 0~~	1 0 0
C	1 0 1	~~1 0 0~~

Finalley, we strikethrough any duplicate test situation

Please note that in the row where C is the determining condition, for (A AND B) any combination can be chosen for which (A AND B) is untrue. So instead of the combination 1-0, the combinations 0-1 or 0-0 could have been chosen.

However, if 0-0 had been chosen one test situation less can be striked through, resulting in n+2 instead of n+1 test situations. The C row would in that case have contained the test situations 0-0-1 and 0-0-0, the latter of which is not a duplicate and can therefore not be striked through.

3.4.2.5 Multiple Condition coverage

All the possible combinations of outcomes of conditions in a decision (therefore the complete decision table) are tested at least once. Since there are only two possible outcomes of a condition (TRUE or FALSE), 2 is the basis for the number of test situations that can be created. The maximum number of test situations (the complete decision table) depends on the amount of conditions: $\mathbf{2^N}$, where **N** is the amount of conditions.

Filling in a table with ones and zeros can be done in many ways. Let's take an easy example with three conditions. This would lead to 23=8 test situations.

We could start by filling the last column with a sequence of 0 1.
For the second column we double the 0's and 1's.
For the first column we again double the 0's and 1's.

A	B	C
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Another handy way of filling in the decision table is with the use of the so called "Gray-code". This causes only one condition to change in value per column.

We now start with the first column. Knowing that we will get 8 test situations, we divide the first column into four 0's and four 1's.
For the second column we spilt the amount of 0's and 1's. But there where it is possible to "mirror" the sequence, we will do so. So in this case after 0011 we will continue with 1100.

For the last column we again split the 0's and 1's. And also here we will mirror the sequence where possible. So after 01 we will continue with 10, and after that we will mirror again and continue with 01 etc.

A	B	C
0	0	0
0	0	1
0	1	1
0	1	0
1	1	0
1	1	1
1	0	1
1	0	0

You can see now that in the second row only value C has changed in comparison with the first row. In the third row only value B has changed in comparison with the second row. Etc. This is helpful for the creation of the physical test cases: copy and paste and change one value.

Multiple Condition Coverage (MCC) can be applied in two ways:

1. All combinations of 0's and 1's of conditions per decision point.
2. All combinations of 0's and 1's of all conditions from all decision points.

The test basis consists of decision tables, pseudo-code, a process description or other (functional) descriptions, in which conditions occur. The conditions and the results are put into a decision table.

- Find conditions in the test basis
- Create a conditions list
- Find results in the test basis and add these to the conditions list
- Fill in the decision table.

3.4.3 Semantics

See section 3.7.5 (Semantic Test).

3.5 Coverage Types Data

3.5.1 Introduction

Data is created and ends when they are removed. In between, the data are used to update them or consult them. The data life cycle can be tested, as well as combinations of input data and the categories of data input or output.

This group consists of the following coverage types:

- Equivalence classes (section 3.5.2)
- Boundary value analysis (section 3.5.3)
- Data Combinations (section 3.5.4)
- Syntax (section 3.5.5)
- CRUD (section 3.5.6)
- Integrity Rules (section 3.5.7)
- Data Flows (not explained separately in this Workbook)
- Domain Testing (not explained separately in this Workbook)
- Right Paths / Fault Paths (not explained separately in this Workbook)

3.5.2 Equivalence classes

Characteristics

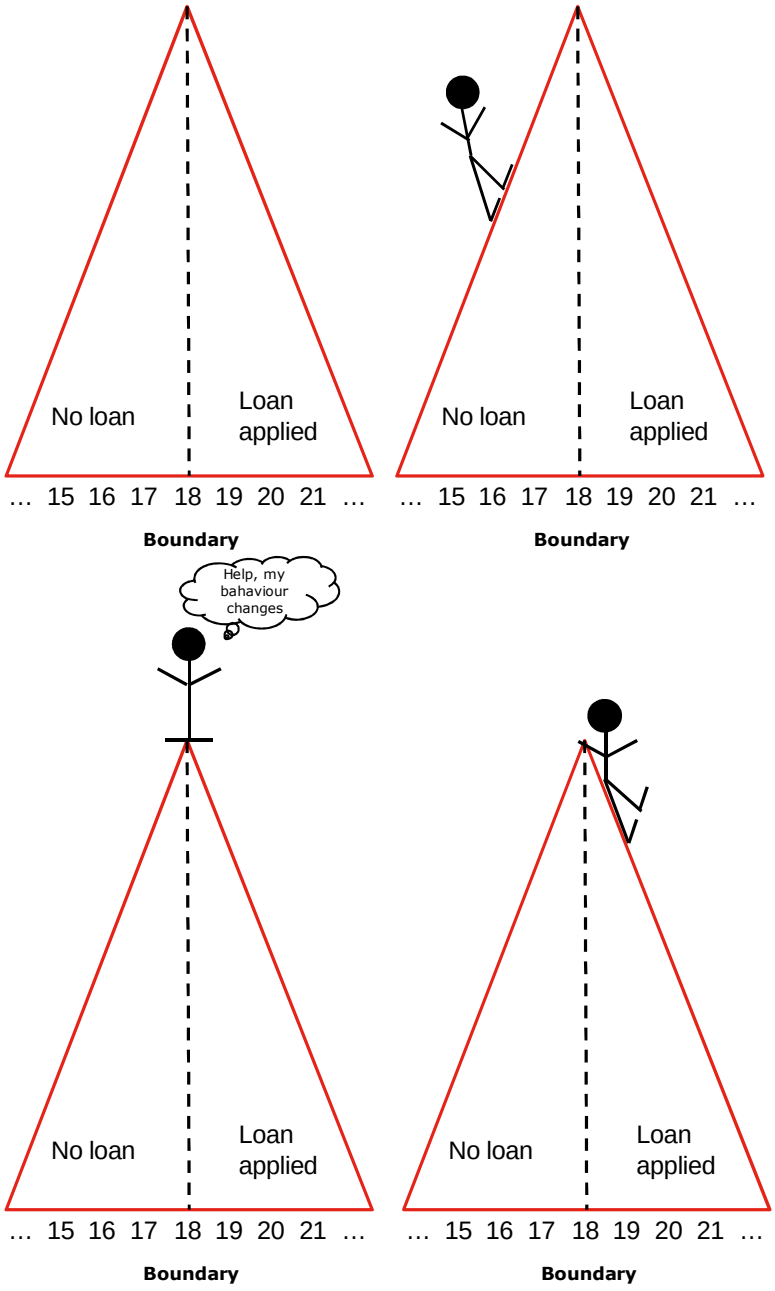
Approach	Coverage based – Data
Quality characteristic / Test Variety	• Primarily functionality • But also applicable to other quality characteristics and test varieties
Test Basis	• Almost all kinds

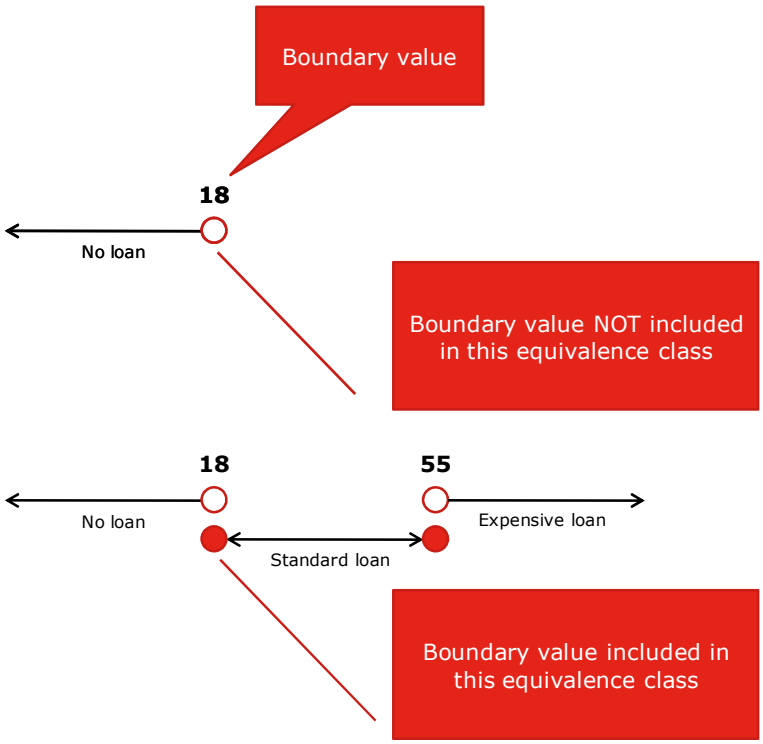
Description

The covering of equivalence classes is a powerful means of achieving a relatively high fault-detection rate with a limited set of test situations. The principle is simple and is applied by most experienced testers automatically and intuitively.

In the application of equivalence classes, the entire value range of a parameter is partitioned into classes, in which the system behaviour is similar (equivalent).

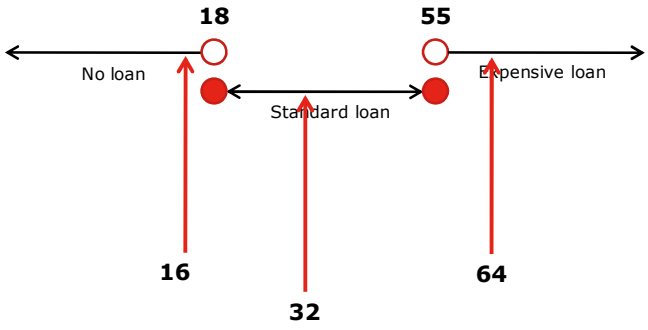
These are called equivalence classes.





The principle behind the application of equivalence classes is that each value taken from a class has the same chance of finding a fault and that testing with several values from the same class barely increases the chances of fault detection. It should be realized that this is an *assumption*. If, with a random value in an equivalence class the correct system behaviour occurs, it is in principle still possible for a fault to occur with another value.

Even though the underlying principle is an assumption, it is a usable and useful one. By basing test cases on these equivalence classes instead of on every possible input value, the number of test cases is restricted, while a satisfactory coverage is obtained of the possible variations in the system behaviour.



Possible values to choose from the equivalence classes

3.5.3 Boundary value analysis

Characteristics

Approach	Coverage based – Data
Quality characteristic / Test Variety	• Primarily functionality • But also applicable to other quality characteristics and test varieties
Test Basis	• Almost all kinds

Description

> *If the system behaviour changes as soon as the value of a parameter exceeds a particular boundary, this is called a ' boundary value'.*

In practice, it appears that many faults are connected with boundaries. Usually these are simply 'sloppy programming mistakes' in which the programmer, for example, has accidentally programmed a ">" instead of a "≥", or a "=" instead of a "≥".

Apart from in equivalence classes, boundaries also often occur in the coverage of conditions and decision points. For example, in the lending system the following condition could be defined:

IF (loan sum > salary) THEN …

Here, the "loan sum" is the parameter with the boundary of "salary".
The testing of whether the boundary values have been allocated to the appropriate equivalence class (or outcome of the condition) is a separate test goal that is achieved by means of " **boundary value analysis**".

The technique for carrying out boundary value analysis is simple in the extreme:

- Determine the boundaries of the relevant equivalence class or condition
- Define the following 3 test situations: exactly on the boundary, directly above it, directly below it.

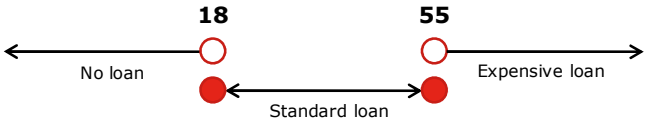
This way of elaborating 3 test situations per boundary value is called the Normal variant. There is also a (S)light variant of boundary value analysis, with which only 2 test situations are tested: the boundary itself plus the adjacent values in the other equivalence class.

If boundary value analysis has not been opted for explicitly, experienced testers will often intuitively apply the slight variant. Indeed, if it is a requirement to test a value from both equivalence classes (above and below the boundary), then "exactly on" and "adjacent to" the boundary value can be selected without any extra effort.

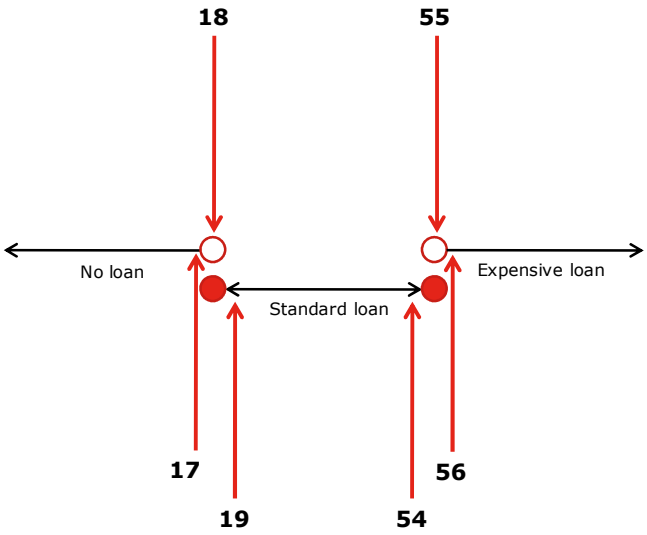
A disadvantage of the slight variant is that this will not uncover certain faults that are found using standard boundary value analysis. An example of this is the previously mentioned fault, in which a "=" has been programmed instead of a "≥".

Boundary value analysis is **not always** applicable to equivalence classes or conditions. Boundaries are not always present. Take, for example, the parameter "Gender" with the values (and therefore equivalence classes) of "M" and "F". There is no such thing as a boundary between the "M" and the "F". This applies also, for example, to all those parameters in a system that belong to 'codes' and 'types'.

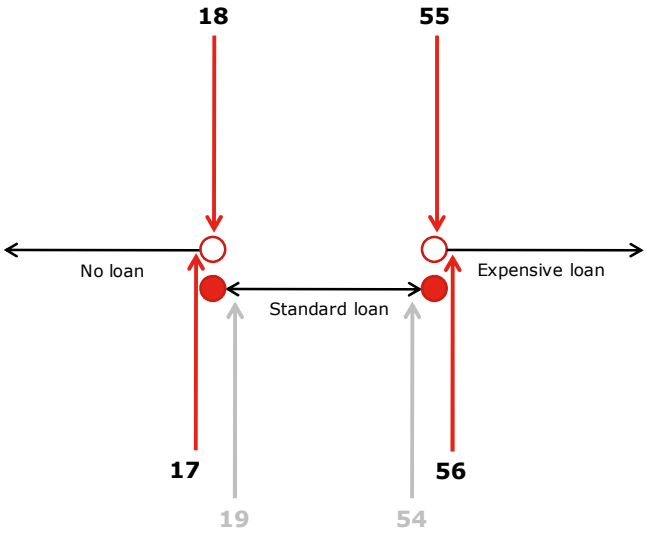
The following is a step-by-step example of the application of boundary value analysis. The example is about a case where a person can get a loan if they are 18 years or older but get a more expensive loan once they are over 55.



18 and 55 are the boundary values



Normal variant
- exactly on the boundary
- directly above it
- directly below it



(S)light variant
- exactly on the boundary
- the adjacent value in the *other* equivalence class

3.5.4 Data Combinations

Characteristics

Approach	Coverage based – Data
Quality characteristic / Test Variety	• Primarily functionality • But also applicable to other quality characteristics and test varieties
Test Basis	• Almost all kinds

Description

Within functionalities single data attributes (and their equivalence classes) may influence the system behaviour, but many times combinations of data attributes are of influence on variations in the system behaviour.

Depending on the agreed test intensity, the different coverage variations can be selected on covering data combinations:

- One or more "data pairs" (testing of the most interesting pairs of data indicated by the experts, e.g. on the basis of risk assessment)
- Pairwise testing
- N-wise testing (extension on Pairwise)
- All possible combinations (= multiple condition coverage applied – see section 3.4.2 – on data, instead of conditions)

3.5.4.1 N-Wise and Pairwise

N-wise testing has the aim of testing all the possibilities of any random combination of N factors.
The maximum value for N is equal to the number of parameters. In that case, the result is equal to the testing of the complete decision table: all the combinations of all the values of all the parameters. In practice, a value of 4 or higher is seldom applied. In order to apply N-wise testing tools are required.

Pairwise testing
The most common application of N-wise testing is pairwise testing. Pairwise testing is based on the phenomenon that most faults in software are the consequence of one particular factor or the combination of 2 factors. The number of faults that are caused by a specific combination of more than 2 factors becomes exponentially smaller. Instead of testing all the possible combinations of all the factors, it is very effective if every combination of 2 factors is tested.

The aim of pairwise testing is to test all the possibilities of any combination of 2 factors.

This delivers an enormous reduction in the number of required test cases, yet still gives a good fault-detection result.

The following example illustrates the meaning of pairwise testing.

In the system under test (for ordering books via the Internet), the following 3 parameters play a role. For each parameter, there are 2 equivalence classes to be tested:

Number of books: Few; many
Sum: Low; high
Membership card: None; Gold card

In order to test all the combinations relating to these 3 parameters, 2x2x2=8 test situations are required, namely::

	Number of books	Sum	Membership card
1	Few	Low	None
2	Few	Low	Gold card
3	Few	High	None
4	Few	High	Gold card
5	Many	Low	None
6	Many	Low	Gold card
7	Many	High	None
8	Many	High	Gold card

For pairwise testing, as few as 4 test situations will suffice, as shown below:

	Number of books	Sum	Membership card
1	Few	Low	None
2	Few	High	Gold card
3	Many	Low	Gold card
4	Many	High	None

Of the 2 parameters [Number of books, Sum], all 4 existing combinations are tested (Few/Low; Few/High; Many/Low; Many/High). The same applies to the other combinations of 2 parameters, so for [Number of books, Membership card] and [Sum, Membership card]. Check it for yourself.

What is the point of this? If a fault exists in the system that occurs when one of the possible values of one of the parameters is combined with a particular value of one of the other parameters, then this fault is always found with these 4 test cases. That is the strength of pairwise testing.

3.5.5 Syntax

See section 3.7.4 (Syntactic Test).

3.5.6 CRUD

Characteristics

Approach	Coverage based – Data
Quality characteristic / Test Variety	• Functionality • Suitability • Connectivity
Test Basis	• CRUD matrix • Functional design and/or • Detailed domain knowledge

Description

Covering all basic operations (Create, Read, Update, Delete) on all entities.

The data that are stored and maintained in the system under test have a *life cycle*. This starts when an entity is created and ends when it is removed. In between, the entity is used by updating it or consulting it.

An overview of the life cycle of the data, or entities, is obtained with the aid of a "*CRUD matrix*". This is a matrix in which the entities are shown horizontally on the axes and the functions vertically. The matrix is filled in using the letters C(reate), R(ead), U(pdate) and D(elete). If a function executes a particular action in connection with an entity, this is shown in the matrix by means of a C, R, U and/or D.

This is illustrated in the figure below:

	Invoice	Article	...
Management of articles	-	C, U, D	...
Create article	C	R	...
Desk payment	C, R, D	-	...
...	...	...	...

Constructing a CRUD-matrix

For the composition of the CRUD-matrix all of the functions in the system are checked. For each identified function is determined:
• which entities are used by this function;
• what actions (C, R, U, and / or D) are performed by these entities.
The result will be entered in the matrix

In more detail

Often, a special structure is visible which has to do with two groups of data elements and functions:

- Master data with management functions.
  Master data are the basic data in the system. For example, "article" and "customer". They are usually maintained independently of the other data with the aid of the management functions linked to them. This generally has the following effect on the CRUD matrix: with a management function, only the column for the relevant master data is completed, and with all the actions: C, R, U and D. If the master data and management functions are defined first (and in the same sequence) in the CRUD matrix, this part of the CRUD matrix will be filled in solely on the diagonal (with "CRUD")
- Derived data with processing functions.
  Derived data are data that are produced by the specific business processes, in which master data is used. For example, "Quote" and "Invoice". It is the processing functions that manage the specific business processes and produce and amend the derived data. This generally has the following effect on the CRUD matrix: processing functions only execute the action "R" on the master data. All the actions can be carried out on the derived data. With the derived data, the management function rows are empty. All the actions (C, R, U and D) are carried out by 1 or more processing functions.

In practice, it is of course permitted to deviate from this, but the reason for doing so would at least require investigation.

The creation of the CRUD matrix is preferably not delayed until during the testing, but it should be delivered as part of the system development by the developer, for the creation of a CRUD matrix is not only useful to testers, but also to the developers themselves: the designing of an information system is usually reasoned from within the functions. Per function, it is described which data will be used. In the creation of a CRUD matrix, reasoning takes place from within the data. Per entity, it is described which functions will use the relevant entity in which way. By creating such a cross-reference table (CRUD matrix) anomalies and/or incomplete areas are sometimes brought to light that would probably not be found with a function-orientated approach, whereas they are now found at an early stage!

Testing the life cycle

The testing of the life cycle consists of 2 parts: the completeness check and the consistency test. These are explained below:

Completeness check

This is a form of *evaluation*, in which it is examined in the CRUD matrix whether all 4 possible actions (C, R, U and D) occur with every entity. In other words, has the entire life cycle been implemented for every entity? The lack of an action does not necessarily mean that the system is wrong. However, the reason for it at least requires investigation.

Consistency test

This is a test that is aimed at integration of the various functions. This checks whether the various functions use an entity in a consistent way. In other words, is the relevant entity being corrupted by one function in such a way that it can no longer be used by the other functions correctly?
Test cases are derived by putting together an entire life cycle of an entity. This is done as follows:

- Every test case starts with a "C", followed by all the possible "U"s and ending with a "D". If there are further possibilities of creating or removing an entity, additional test cases are designed
- After every action (C, U or D), an "R" is carried out once or more. This is to establish that the entity has been correctly processed and is usable for the other functions (has not been corrupted)
- In respect of the relevant entity, all the occurrences of actions (C, R, U and D) in all the functions should be covered by the test cases.

With this, CRUD is fully covered in principle.

More thorough coverage of CRUD can be achieved by requiring that combinations of actions also be fully covered. For example, by requiring that after each "U" *all* the functions with an "R" should be carried out.

The example below illustrates this:

In more detail

Suppose that the entity "Order" is processed as follows by the following functions:

Create order (C); Cancel order (D); Part-delivery (U); Overview of orders (R); Stock control (R)

The standard coverage of CRUD is then achieved with the following test case:

Create order	(C)	
Stock control	(R)	
Part-delivery	(U)	
Overview of orders	(R)	
Cancel order	(D)	
Overview of orders	(R)	

However, with this, the following fault would not be found: after a partial delivery, the stock control is no longer correct, because it is (wrongly) treating the whole order as having been delivered. This fault would have been found with the more thorough variant, which gives the result with the following test case:

Create order	(C)	
Overview of orders	(R)	
Stock control	(R)	
Partial delivery	**(U)**	**(causes fault in "Stock control")**
Overview of orders	(R)	
Stock control	**(R)**	**(fault is found)**
Cancel order	(D)	
Overview of orders	(R)	
Stock control	(R)	

3.5.7 Integrity rules

Characteristics

Approach	Coverage based – Data
Quality characteristic / Test Variety	• Overarching functionality • Suitability • Connectivity
Test Basis	• Description of integrity rules, • Functional design and / or • Detailed domain knowledge

Description

Integrity rules describe the preconditions under which certain CRUD processes are or are not permitted.

For example, "Entity X may only be changed if the linked entity Y is removed from it". Besides this, functional specifications or detailed domain expertise is necessary in order to be able to predict the result of each test case.

The coverage of Integrity rules has a strong relationship with the coverage type CRUD (Create, Read, Update, Delete). They can very well be applied together.

Since the integrity rules can be described as decision points, decision coverage can be applied.

The following activities should be carried out:

- Gather the integrity rules on the selected entities. These are the rules that define under which conditions the processing of the entities is valid or not. Integrity rules are usually specified within the functional specifications, database models or in separate business rules.
- Apply decision coverage (see section 3.4.2). This means that for each integrity rule, two test situations are derived:
  - Invalid: The integrity rule is not met. The process is invalid and should result in correct error handling.
  - Valid: The integrity rule is met. The process is valid and should be executed.

Example

A payment agreement may not be removed as long as there is an outstanding invoice with the relevant payment agreement. This leads to two test situations:

- *IR1-1: Delete (D) payment agreement, while an invoice is outstanding with the relevant payment agreement*
- *IR1-2: Delete (D) payment agreement, without there being an outstanding invoice with the with the relevant payment agreement*

A shortened clear notation for this type of test situation is, for example:

Test situation	Action	Entity	Condition	Valid Y/N
IR1-1	D	payment agreement	outstanding invoice	N
IR1-2	D	payment agreement	no outstanding invoice	Y

The abbreviation "IR" means "Integrity Rule".

3.6 *Coverage Types Appearance*

3.6.1 Introduction

How a system operates, how it performs, what its appearance should be, is often described in non-functional requirements.

Coverage types in this group are:

- Presentation (section 3.6.2)
- Load profiles (section 3.6.3)
- Operational profiles (section 3.6.4)
- Heuristics (section 3.6.5)

Other coverage types in this group are (not described in this Workbook):

- Information security-based:
  - Authorization
  - Authentication
  - Communication security
  - Data confidentiality
  - Data integrity
  - Non-repudiation
  - Privacy

- Usability-based:
  - Alpha-testing
  - Beta-testing
  - Usability lab

3.6.2 Presentation

See section 3.7.4 (Syntactic Test).

3.6.3 Load Profiles

Characteristics

Approach	Coverage based – Appearance
Quality characteristic / Test Variety	• Performance • Continuity • Connectivity • Effectivity
Test Basis	• Description of realistic use (profiles)

Description

Load profiles describe the loading under which the system operates in terms of how many users are operating the system at once. The testing of load profiles has the aim of examining whether: "The system still works correctly when *many* transactions are carried out by many users at once".

Load profile are often applied in combination with coverage type Operational profiles.

Load profiles show the degree to which the system resources (CPU, memory, network capacity) are loaded in reality. The loading is usually shown in terms of the number of users or number of times that a transaction is carried out in a particular period. Usually, the loading of a system is not continuously even, but varies over a period of time: there are peaks and valleys within a 24-hour stretch. Often, weekends will show a different loading from weekdays. And during holiday periods and public holidays, the loading of a system may look different again.

For the creation of a load profile, information from the following sources is combined:

- Measuring the loading of the system using specific tools (monitors). The responsibility for this usually resides with a department for "Technical System Administration".
- Interviewing users. In fact, this amounts to the following questions: "Which transactions do you carry out? How often, and when?"

The testing of load profiles comes under what is often referred to as "performance testing" and is a testing specialism in itself. While it is possible to do manually, tools are usually employed that generate a particular loading of the system. Using the tools, a realistic loading is *simulated*, such as:

- Creation of virtual users.
  A virtual user is a small program that simulates a user. On one PC, many such programs can run at once. This avoids the need for the physical presence of a separate PC for each user. This is mainly applied for subjecting the entire system, including the network, to a particular loading.
- Offering transactions via the database-management interface.
  This creates a certain loading of the back-end of the system without overloading the front-end or the network. It facilitates direct measurement of whether the database server has the appropriate dimensions.

There are various types of performance tests that each have a different goal. The most common are:

- Testing with normal or average usage.
  The aim here is to examine whether the available system resources are adequate for the 'usual' circumstances. The idea here is, that it can be commercially advantageous to deploy extra resources for the rare occasions that 'exceptionally heavy loading' takes place.
- Testing with peak loading.
  The aim here is to examine whether there are sufficient system resources for even the most demanding circumstances that may arise in practice.
- Measuring the breaking point.
  This is also known as "stress testing". The aim here is to examine what the maximum load is under which the system will still perform acceptably. With a particular system configuration, the loading is stepped up, while the response time is measured. This can be shown in a graph. At the point when the graph shows a sharp incline, the response time increases disproportionately fast (the response 'collapses') and the breaking point has been reached.

3.6.4 Operational Profiles

Characteristics

Approach	Coverage based – Appearance
Quality characteristic / Test Variety	• Performance • Continuity • Connectivity • Effectivity
Test Basis	• Description of realistic use (profiles)

Description

An operational profile describes in quantitative terms how the system is used by a particular type of user. This concept was introduced by John Musa; you are referred to his work [Musa, 1998] for a more comprehensive discussion of operational profiles. Below is a brief explanation.

An operational profile describes the realistic usage by answering the question: "When the system is in *this condition*, how great is the *chance* that *this action* will be carried out by the user?" In the literature, instead of *condition* and *action*, reference is usually made to *history class* and *event*. An operational profile provides a statistical average of how 'the user' handles the system. If various types of users are distinguishable who display significantly varied statistical average behaviour, it is advisable to create a separate operational profile per user type.

Operational profiles are often applied in combination with coverage type Load profiles.

3.6.5 Heuristics

Characteristics

Approach	Coverage based – Appearance
Quality characteristic / Test Variety	• User-friendliness • Suitability • Effectivity • Usability
Test Basis	• Description of realistic use (profiles)

Description

Heuristic evaluation is one of the best-known ways of testing usability. During a heuristic evaluation, a systematic examination is carried out of the usability of the design of the user interface. The ultimate aim of heuristic evaluation is to discover problems in the design of the user interface. By finding such problems at the design stage, it is possible to solve them in time. During the process of heuristic evaluation, a group of 3-5 experts (evaluators) give their opinion on the user interface in accordance with a number of usability principles (also known as the "heuristics").

In more detail

Nielsen distinguishes 10 heuristics; see [Nielsen, 2006]:

- Visibility of the system status
- Match between the system and the real world
- User control and freedom
- Consistency and standards
- Error prevention
- Recognition rather than recall
- Flexibility and efficiency of use
- Aesthetic and minimalist design
- Help for users to recognize, diagnose and recover from errors
- Help and documentation

3.7 A basic set of test design techniques

3.7.1 Introduction

The basic principles of test design techniques are described in section 3.2.2 (The Generic Test Design Steps) and section 3.2.5 (Test design techniques).
In this section, a number of test design techniques will be explained:

- Data Combination Test (section 3.7.2)
- Process Cycle Test (section 3.7.3)
- Syntactic Test (section 3.7.4)
- Semantic Test (section 3.7.5)
- Decision Table Test (section 3.7.6)

- Elementary Comparison Test (section 3.7.7)
- Data Cycle Test (section 3.7.8)
- Real Life Test (section 3.7.9)

3.7.2 Data Combination Test (DCoT)

Characteristics

Approach	Coverage based – Data But also: Experience based
Quality characteristic / Test Variety	• Overarching functionality • Detailed functionality
Coverage Type	• Equivalence classes • Optional: data combinations • Optional: boundary value analysis
Test Basis	• All types of information on the functionality of the system • Including: domain expertise

Description

The data combination test (DCoT) is a versatile technique for the testing of functionality both at detail level and at overall system level. In the embedded world, this technique is also known as the "Classification Tree Method". It was developed by Grochtmann and Grimm, and is described in "Testing Embedded Software" [Broekman, 2003] and elsewhere.

For the DCoT, no specific test basis is required. All types of information on the functionality of the system are usable:

- Formal system documentation, such as functional design, logical data model and requirements,
- Informal documentation, such as manuals, folders, pre-surveys and memos,
- Domain expertise that is not documented, but resides 'in the experts' heads'.

The fact that domain expertise is usable as a test basis also makes this technique suitable for situations in which specifications are incomplete or out of date, or even unavailable altogether.

Because of the strongly informal character of this technique, the quality of the test cases designed with it is largely determined by the expertise and skill of those involved. For that reason, the DCoT is preferably carried out by a team of 2 to 5 persons with a mix of expertise: test, domain and system expertise.

Tip

Organize a 'creative session', such as brainstorming or meta-planning, in which the tester acts as moderator of the process. Invite one expert to this session from every relevant discipline, e.g. a user, an administrator and a system developer or system architect. The experts will supply the substantive information, which can be structured by the tester and converted into test situations and test cases.

With the DCoT, the test situations are determined by reasoning from within the data attribute as to which variations should be tested. The coverage type that is always used here is:

- Equivalence classes.

Depending on the agreed test intensity, the coverage can be extended by fully combining the equivalence classes of two or more different data. For this coverage type *data combinations* is used:

- One or more "data pairs" (testing of the most interesting pairs of data indicated by the experts, e.g. on the basis of risk assessment)
- Pairwise testing
- N-wise testing (extension on Pairwise)
- All possible combinations (= multiple condition coverage applied – see section 3.4.2 – on data, instead of conditions)

Besides these, there is the option of reinforcing the test by applying *boundary value analysis*. This can also be applied selectively, by defining the boundary values for specific data attributes as a separate equivalence class.

Thanks to its versatility, the data combination test is suitable both for testing those functions that are deemed very important, and for testing system parts that 'just need a quick test'.

Points of focus in the steps

In this section, the data combination test is explained step by step. The generic steps of test design (see section 3.2.2) are the starting point. Every step of this technique is explained by means of the same example case.

Example case

This example is about a function with which reservations can be made for a flight:
The user enters a number of data on the composition of the group (adults, children, infants) as well as on the planned journey (destination, period).
After that, the user can choose the criteria by which the most suitable flight should be searched.
The system will then show a list of possible flights *or* a message, in case there is no flight available that meets the criteria and has the needed number of seats still available.
This function must be tested with average test intensity, using the DCoT.

1- Identifying test situations

Identifying test situations is the creative step in the process and is ideally carried out by a team in which various forms of expertise are represented. During this step, the following activities are carried out:

- Determine the data attributes that influence the functionality. This does not automatically mean all the data attributes that are used by the function. It concerns the data attributes that are of influence on variations in the system behaviour. This includes the data attributes for which equivalence classes can be determined. The defined data can consist of entities, attributes or functional concepts in a general sense..
- Determine the equivalence classes for each data attribute.
  See section 3.5.2 for this.
- Determine the relationships between the data attributes.
  Some data attributes are only of influence on the system behaviour under certain conditions, namely if another data attribute has a value from a specific equivalence class. That means that the possible variations of the first-mentioned data attribute must be combined with the specific value of the last-mentioned data attribute. In the

example set out, such a relationship is visible between the data attributes "search criterion" and "flies to that destination".

The result can be illustrated in a 'classification tree':

- Data attributes that logically belong together can be grouped under an overall title, such as "personal details" or "employer types"
- Under every data attribute the equivalence classes are hung, like branches on the tree
- Relationships between the data can be shown simply by hanging the relevant parts of the classification tree directly under the relevant equivalence class.

The creation of the classification tree with which the test situations are identified is an iterative and interactive process, in which the parties involved inspire, correct and complement each other. How far this process will go is the choice and responsibility of the team. A test manager who wishes to keep this well under control will provide a concrete job description for the team and request regular feedback on the results.

If required, it is defined which data attributes are eligible for 'fully combined testing'. That means that all the possible combinations of all the equivalence classes of those data attributes should be tested. How many of such combinations should be defined depends on the agreed test intensity.

Tip

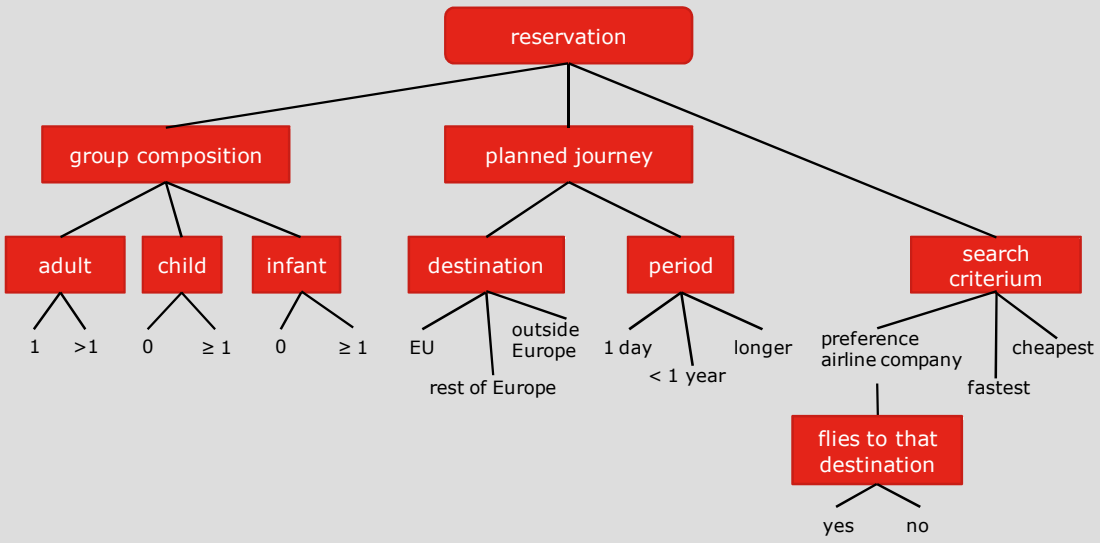
The following can be used as a rule of thumb:

Light	No or just one data pair.
Average	Two or more data pairs. This offers an increasing scale of test intensity that ends with "pairwise testing".
Thorough	Average test intensity + boundary value analysis.

Instead of combinations of two data attributes (data pair) it can be defined that all combinations of three data attributes (data triplet) must be tested. This implies an increase of the test intensity.

Example case solution

For the function "seat reservation" the team came up with the following classification tree:



The following aspects have been taken into account:

- A passenger can either be an adult, a child or an infant. An infant has no seat of its own on the plane.
- A planned journey that is longer than a year might cause confusion on the date for the homeward journey.
- An airline does or does not fly to the planned destination. This is only relevant under the assumption that this airline has been entered as a search criterion.

Fitting the agreed average test intensity, two data pairs have been defined that must be tested fully combined:

- child – infant (4 mandatory combinations)
- destination – flies to that destination (6 mandatory combinations)

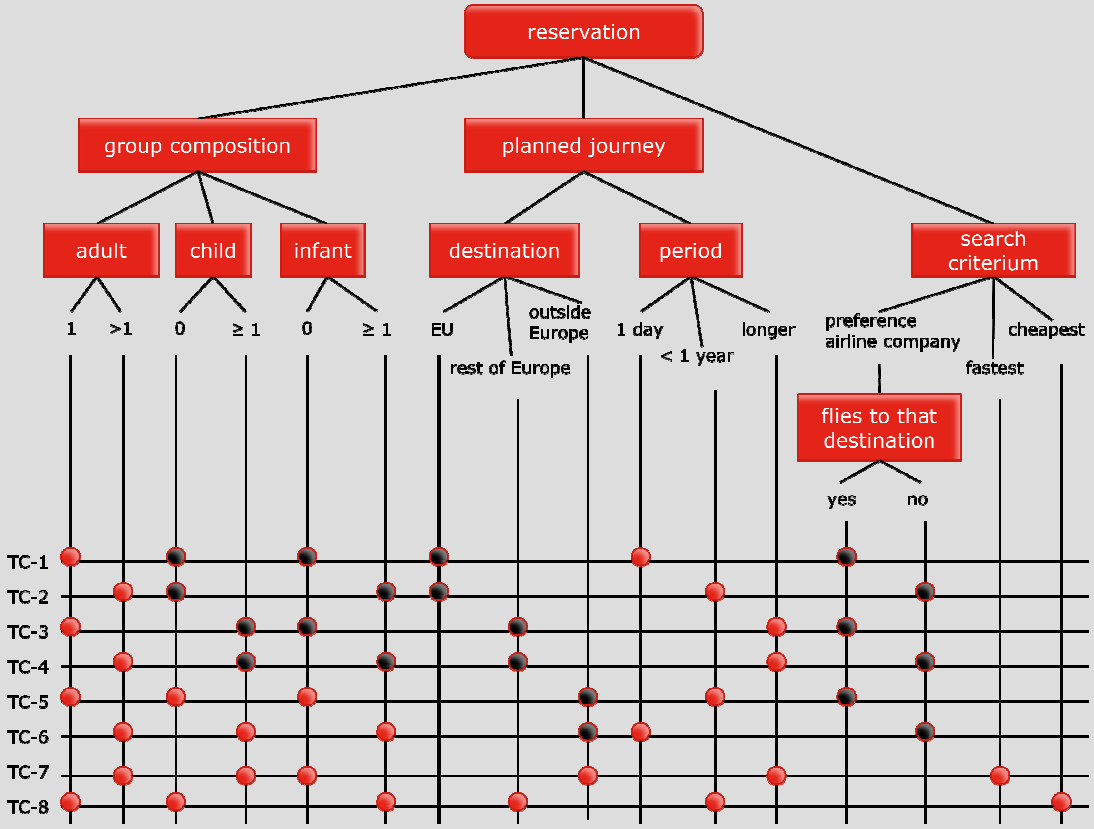
2- Creating logical test cases

With a logical test case, precisely *one* of the equivalence classes is covered for *every* data attribute in the classification tree. Collectively, the logical test cases should at any rate cover *all* the equivalence classes of all the data attributes. Depending on the chosen test intensity, they should also cover *all* the combinations of equivalence classes of particular data attributes, if necessary. Basically, there are two ways of demonstrating this clearly:

- In table form.
  This method is usually employed where the "pairwise testing" (see section 3.5.4) option has been adopted. Tools for pairwise testing normally deliver their results directly in table form.
- Graphic depiction of a 'classification tree'.
  This is particularly useful if the most elementary form of testing (without combinations) has been chosen, or for the selective application of "complete decision table" coverage. Ideally, a graphic tool should be used here. The tool "Testona" can be downloaded via the Internet (http://www.berner-mattner.com/de/produkte/testona/index.html).

Example case solution

The logical test cases have been depicted in the classification tree:



The two data pairs that must be tested fully combined have been taken into account first (the black pellets). Test cases TC-1 to TC-4 cover the data pair "child - infant", whilst the data pair "destination – flies to that destination" are covered by the test cases TC-1 to TC-6. The next steps are to ensure that every equivalence class is covered at least once and to make every test case complete so that for every test case every relevant data attribute has a value (the red pellets).

Note: For achieving minimal coverage (only covering the equivalence classes and not any combinations of data pairs), 4 logical test cases will suffice here. For instance TC-1, TC-4, TC-7 and TC-8.

3- Creating physical test cases

In creating the physical test cases, concrete values should be chosen for all the input data. These input data do not always correspond exactly with the concepts maintained in the classification tree. For example, the classification tree may contain the concept of "duration", while the function to be tested expects the data "Start date" and "End date".

Every physical test case should have a concrete predicted result. However, this generally depends on the other data and system settings that belong with the chosen starting point.

Example case solution

To illustrate this, in the table below four of the test cases have been made physical. For each test case the physical values of all needed input data have been defined. Furthermore, the predicted result has been made concrete.

	TC-1	**TC-2**	**TC-3**	**TC-7**
customer name	O'Brien	Smith	Atkinson	Cleese
#adults	1	3	1	2
#children	0	0	2	1
#infants	0	1	0	0
destination	France-CdG	Germany-Frankfurt	Switzerland-Zürich	Singapore-Changi
date of departure	12-02-2016	14-02-2016	15-02-2016	16-02-2016
date of return	12-02-2016	15-02-2016	16-02-2017	23-02-2017
search criterion	KLM	Kenya Airways	Canada Air	Fastest
Predicted result		Message: "Airline does not fly to destination of choice."		
airline			Canada Air	Singapore Airlines
flight number	KL1288		CA0833	SA0455
price	€ 144		€ 283	€ 956

In order to predict the results of every physical test case, it is necessary to know exactly which flights and prices are stored in the database. This step goes hand-in hand with the next step, "Establishing the starting point".

4- Establishing the starting point

No remarks.

Example case solution

The following database has to be loaded:"TST_RES_03". This contains in particular the situation that the company "Senegal Airlines" exists, but does not recognize "Eindhoven Airport" as a destination.

Set the system date to 01-02-2006(1 February 2006).

3.7.3 Process Cycle Test (PCT)

Characteristics

Approach	Coverage based – Process
Quality characteristic / Test Variety	• Suitability • Functionality
Coverage Type	• Paths: test depth level 2

Test Basis	• AO description
	  • procedure diagram
	  • description business processes / work processes (like workflows)
	• Functional specifications

Description

The process cycle test is a technique that is applied in particular to the testing of the quality characteristic of Suitability (integration between the administrative organization and the automated information system). The test basis should contain structured information on the required system behaviour in the form of paths and decision points. The process cycle test digresses on a number of points from most other test design techniques:

- The process cycle test is not a design test, but a structure test: the test cases issue from the structure of the procedure flow and not from the design specifications.
- The predicted result in the process cycle test is simple: the physical test case should be executable. This checks implicitly that the individual actions can be carried out. In contrast to other test design techniques, no explicit prediction is made of the result, and so this does not have to be checked.

The process cycle test focuses on the coverage of the variations in the processing. The coverage type used in this is:

- Paths: test depth level 2

Variations on the process cycle test can be created by applying variations of the coverage type:

- Paths: test depth level 1, test depth level 3 and higher
  With this, paths can be tested with respectively less or more depth.

Points of focus in the steps

In this section, the process cycle test is explained step by step. The generic steps of test design (see section 3.2.2) are the starting point. Every step of this technique is explained by means of the same example that was used to explain coverage type paths (see section 3.3.2).

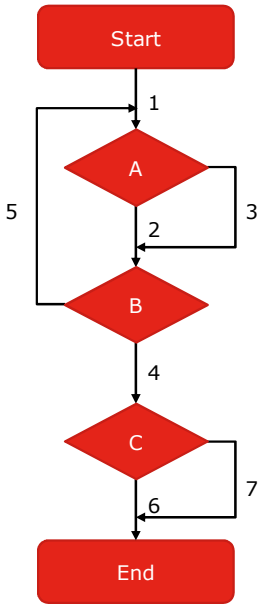
1- Identifying test situations

In order to apply the process cycle test, a process diagram is required. This diagram should contain, besides a start and end point, decision points and paths. If the test basis already contains a diagram, then for the sake of clarity it is often useful to 'undress' it, so that it only contains the above-mentioned aspects. If there is no diagram present in the test basis, the tester will have to distil the decision points and paths from the test basis himself in order to

create a diagram. Subsequently, the test situations are derived from the diagram using coverage type paths: test depth level 2, as described in section 3.3.2.

1- Identifying test situations

The first step is identifying the path combinations as seen in coverage type Paths (test depth level 2)



A: IN: 1,5
OUT: 2,3

B: IN: 2,3
OUT: 4,5

C: IN: 4
OUT: 6, 7

Path combinations - test depth level 2				
A	1-2	1-3	5-2	5-3
B	2-4	2-5	3-4	3-5
C	4-6	4-7		

2- Creating logical test cases

The creation of the logical test cases consists of two activities:

- Creating a set of logical test cases
- Describing the consecutive actions per logical test case.

In the creation of the set of logical test cases, all the test situations should be covered. A test case is defined by going through the process in a certain way from "Start" to "End". The tester is free to choose the way in which the process is followed, as long as all the test situations are covered *at least once*.

If necessary, it can be shown with the aid of a cross-reference matrix that all the test situations are covered with the chosen set of test cases.

Basically, there are two ways to create a covering set of test cases:

- Working from the process chart, define a test case by running through the process in a particular way from "Start" to "End". The tester is in principle free to choose the exact way of going through the process. Cross out of the list of path combinations every combination that occurs in this test case. Repeat this process until the list of path combinations is completely crossed out.

- Working from the list of path combinations, start with a path combination that begins at "Start". Seek a subsequent path combination that starts with the path with which the previous one ends – like setting down domino tiles, in fact. Continue seeking a subsequent path combination until "End" has been reached. Obviously, previously unused path combinations should be used as much as possible

2- Creating logical test cases
- Creating a set of logical test cases

Go through the process from "Start" to "End", until every test situation has been covered at least once by a logical test case.

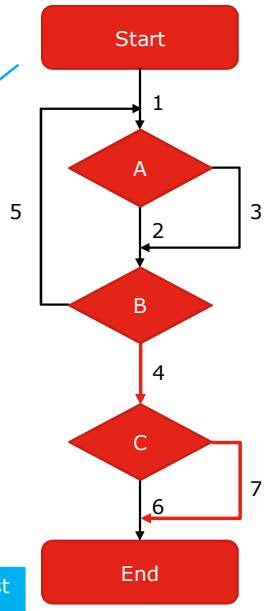
A: IN: 1,5
OUT: 2,3

B: IN: 2,3
OUT: 4,5

C: IN: 4
OUT: 6, 7

Path combinations – level 2				
A	~~1-2~~	1-3	5-2	~~5-3~~
B	2-4	~~2-5~~	~~3-4~~	3-5
C	4-6	~~4-7~~		

Logical test cases	
TC 1	1-2-5-3-4-7



Continue this until all test situations (path combinations) have been covered.

2- Creating logical test cases
- Creating a set of logical test cases

Go through the process from "Start" to "End", until every test situation has been covered at least once by a logical test case.

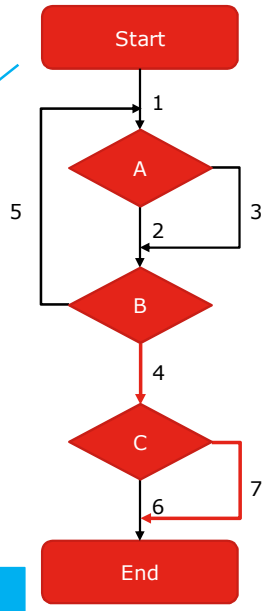
A: IN: 1,5
OUT: 2,3

B: IN: 2,3
OUT: 4,5

C: IN: 4
OUT: 6, 7

Path combinations – level 2				
A	~~1-2~~	~~1-3~~	~~5-2~~	~~5-3~~
B	~~2-4~~	~~2-5~~	~~3-4~~	~~3-5~~
C	~~4-6~~	~~4-7~~		

Logical test cases	
TC 1	1-2-5-3-4-7
TC-2	1-3-5-2-4-6



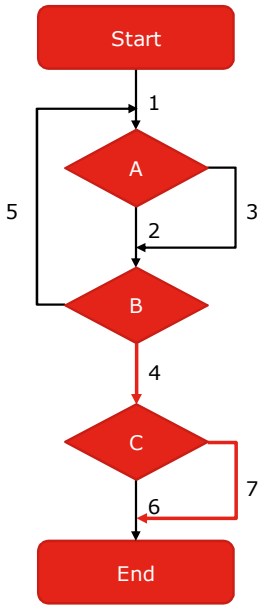
Continued until all test situations were covered.

The logical test cases can then be written out. This means that for each test case a row of consecutive actions is described, in such a way that the execution of these actions will touch on all the test situations from the test case. This activity requires inventiveness and is therefore rather difficult to describe in general terms.

2- Creating logical test cases

- Describing the consecutive actions per logical test case

Logical test cases	
TC 1	1-2-5-3-4-7
TC 2	1-3-5-2-4-6



TC1 (1-2-5-3-4-7):

- •A1-1 Create claim form (Insured)
- •A1-2 Enter claim form details into the system (incomplete) (Employee)
- •A1-3 Start the process "Check for completeness" (Employee)
- •A1-4 Contact the insured party to complete the details (Employee)
- •Etc.

3- Creating physical test cases

Besides the previously mentioned differences from the other test design techniques, there is another difference to note. With the test execution, there is more required in the process cycle test than just the technical test infrastructure on which the automated part of the information system runs. The manual procedures mainly have to be carried out by various types of employees, which means that several testers are required to play particular roles in the test execution. It is of course also possible to have the test executed by one tester who possesses several user IDs, repeatedly logging in and out during the test execution. In addition, the required data are only partly present in the database of the automated part of the information system, and the rest is outside the system, for example in the form of completed forms. That, too, is different from the other test design techniques.

In the creation of the physical test cases, a physical formulation of the logical test cases is supplied. With this, the actions described serve as a starting point.

3- Creating physical test cases

TC1 (1-2-5-3-4-7):

Insured party

•A1-1 Create claim form with following details:
- •Name : Smith, J.
- •Etc.

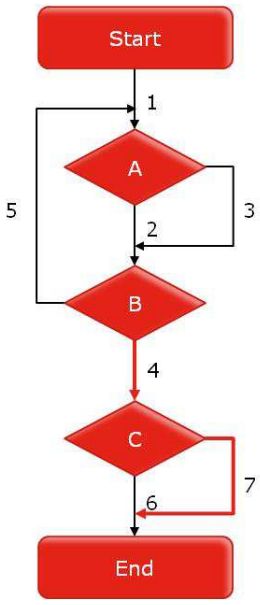
Employee

•A1-2 Enter claim form details into the system (without "Date of loss" and without "Description of loss")

•A1-3 Start "Check for completeness" process (form is incomplete)

•A1-4 Contact insured party to obtain "Date of loss"

•Etc.



4- Establishing the starting point

No remarks.

5- Creating the test script

Often not done, since the physical test cases already consist of full scenarios.

3.7.4 Syntactic Test (SYN)

Characteristics

Approach	Coverage based – Data Coverage based – Appearance
Quality characteristic / Test Variety	• Functionality (Validation test) • User-friendliness (Presentation test)
Coverage Type	• Syntax • Presentation
Test Basis	• Data dictionary or other data models • Style guides • List and screen specifications

Description

The syntactic test, together with the semantic test, belongs to the *validation tests*, with which the *validity of the input data* is tested. This establishes the degree to which the system is proof against invalid, or 'nonsense' input that is offered to the system willfully or otherwise. This test is also used to test the validity of *output data*.

Besides that, the syntactic test is also a *presentation test*, which tests the layout of the screens. Presentation tests can be applied to both input (screens) and output (lists, reports).

3.7.4.1 Validation test

Validation tests focus on *attributes*, which should not be confused with *data*. An input screen or other random interface contains attributes that are (to be) filled with input values. If the sections contain valid values, the system will generally process these and create or change certain data within.

The test basis for the syntactic test consists of the syntactic rules, which specify how a attribute should comply in order to be accepted as valid input/output by the system. These rules actually describe the value domain for the relevant attribute. If a value outside this domain is offered for the attribute, the system should discontinue the processing in a controlled manner – usually with an error message.

Syntactic rules may be established in various documents, but they are usually described in:

- The 'data dictionary' and other data models, in which the characteristics of all the data are described
- Functional specifications of the relevant function or input screen, containing the specific requirements in respect of the attributes.

The syntactic rules may take a random order and be tested independently of each other.

Usually, in practice, the input screens of data are used to test the syntactic checks. Coverage type Syntax is being used.

Overview of attribute checks:

- Data type
  E.g. numeric, alphabetical, alphanumeric, etc.

- Field length
  The length of the input field is often limited. Investigate what happens when you attempt to exceed this length. (Press the letter key for some time, for instance.)

- Input / Output
  There are 3 possibilities here:
  - I: No value is shown, but may be or must be entered
  - U: The value is shown, but may not be changed
  - UI: A value is shown, and may be changed.

- Default
  If the attribute is left empty, the system should process the default value.
  If it concerns a UI field (see above), the default value should furthermore be shown.

- Mandatory / Non-mandatory
  A mandatory attribute may not remain empty.
  A non-mandatory attribute may remain empty. In the processing, either the datum is left empty or the default value for this datum is used.

- Selection mechanism

A choice has to be made from a number of given possibilities. It is important here whether only one possibility may be chosen or several. This is particularly the case with GUIs (Graphical User Interface), e.g. with:
    - Radio buttons (try to activate several)
    - Check boxes (try to activate several)
    - Drop-down box (try to change the value or make it empty).

- Domain
  This describes all the valid values for this attribute. It can, in principle, be shown in two ways:
    - Enumeration
      For example {M, F, N}
    - Value range
      All the values between the given boundaries are permitted. The value boundaries themselves, in particular, should be tested. For example, [0, 100>, where the symbols indicate that the value range is from 0 to 100, including the value 0, but excluding the value 100.

> Tip
>
> In practice, the value 0 (zero) can cause problems in input fields. It is advisable to try out the value 0 at every input field.

- Special characters
  Can the system handle special characters, such as quotes, exclusive spaces, question marks, Ctrl characters, etc.?

- Format
  For some attributes, specific requirements are set as regards format, e.g.:
    - Date
      Common formats, for example, are YYYYMMDD or DD-MM-YY
    - Postal code
      The postal code format basically varies from country to country. In the Netherlands, the format for this is "1111 AA" (four digits followed by a space and two letters).

3.7.4.2 Presentation test

Presentation tests test the layout. They can be applied to both input (screens) and output (lists, reports). Coverage type Presentation is being used.

Overview of format checks:

- Headers / Footers
    - Are the standards being met in this regard?
      For example: standards for screen or list name, system or print date, version number.
- Attributes
    - Per attribute, specific formatting requirements are defined.
      For example: name of the attribute, position of the attribute on the screen or overview (like the position of the address when the letter is being sent in a window envelope) or depiction of the attribute, such as font, color, etc.
- Other screen objects

- If necessary, such checks as are carried out on "Attributes" can be applied to other screen objects, such as "push buttons" and "drop-down lists".

3.7.5 Semantic Test (SEM)

Characteristics

Approach	Coverage based – Conditions
Quality characteristic / Test Variety	• Functionality (Validation test)
Coverage Type	• Semantics => Decision points: Modified Condition / Decision Coverage
Test Basis	• Functional specification • Overarching 'business rules'

Description

The semantic test (SEM), together with the syntactic test, belongs to the *validation tests*, with which the *validity of the data input* is tested. In practice, the semantic test is often executed in combination with the syntactic test (see section 3.7.4).

The test basis consists of the semantic rules that specify what a datum should comply with in order to be accepted by the system as valid input. Semantic rules are about the *relationships between data*. These relationships may be between the data within a screen, between data on various screens and between input data and existing data in the database. Semantic rules may be established in various documents, but are usually described in:

- Functional specifications of the relevant function or input screen
- The business rules that apply to the functions overall

Tip

If the semantic rules describe the conditions for meeting security requirements, the SEM can also be applied to the test type "Security test".

With the semantic test, user-friendliness aspects can also be tested, by assessing the messages that occur in invalid situations:

- Are they understandable and unambiguous?
- Do they offer clear indications of how the invalid situation can be resolved?

Since the semantic rules can be specified as decision points that consist of compound conditions, for the semantic test one of the coverage types from the area of decision points is selected. The default choice for the semantic test is:

- modified condition/decision coverage.

Variants can be realized simply by replacing this with:

- condition/decision coverage, for a lighter variant
- multiple condition coverage, for a more thorough variant.

Points of focus in the steps

In principle, for the SEM, too, the generic steps (see section 3.2.2) are carried out. However, the construction of a semantic test is very simple: each semantic rule is tested

separately. Each rule leads to one or more test situations and each test situation generally leads to one test case.

For that reason, this section is restricted to explaining the first step "identifying test situations". This will be explained and expanded on through an example.

1- Identifying test situations

A semantic rule that describes the conditions of validity can generally be set out as follows:

IF (semantic rule)	THEN	valid input or processing
	ELSE	error message

In the event that the semantic rule describes the *in*valid situations in which an error message should occur, this becomes:

IF (semantic rule)	THEN	error message
	ELSE	valid input or processing

The semantic rule is a decision point that consists of one or more conditions connected by AND and OR. A single condition has only two test situations, one for valid input and one for invalid input. For compound conditions, the test situations are derived by applying modified condition/decision coverage (MCDC), as explained in section 3.

Example

Suppose that the following semantic check is specified:

"IF customer lives in the Netherlands AND (postal code does not comply with Netherlands format OR country code is different from 31) THEN this results in an error message."

The following occurs after applying MCDC:

D1 A AND (B OR C)	1 error message	0 valid input
A: customer in NL	1 1 0 (1)	0 1 0 (3)
B: postal code not in NL	~~1 1 0~~	1 0 0 (4)
C: country code ≠ 31	1 0 1 (2)	~~1 0 0~~

In more detail

In practice, semantic rules are sometimes described in the form:

"IF item X meets condition A, THEN condition ... should also be met"

The pitfall here is that it appears as though the semantic rule only consists of the condition "IF item X meets condition A". However, that is not the case. Everything that comes after the "THEN" also describes the conditions that should be met. In fact, this way of writing the semantic rule is an example of the "imply operator" in Boolean algebra. The truth table for this operator, which is shown by the symbol "➔", is:

A	B	A ➔ B
1	1	1
1	0	0
0	1	1
0	0	1

Now, a condition that is described by the imply operator can be converted simply into a compound condition with the same truth table:

"A ➔ B" is equivalent to "(NOT A) OR B"

Coverage type modified condition/decision coverage can be applied to the resulting compound condition – that contains only the operators AND, OR and NOT – without difficulty.

The example below explains this further.

Suppose that the following semantic rule is specified:

"When code_contribution = V THEN code_employment must be = F AND Age ≥ 55"

An imply operator has been applied here, whereby the rule actually looks like this:

"code_contribution = V ➔ (code_employment = F AND Age ≥ 55)"

This can be converted into the following compound condition:

"(NOT code_contribution = V) OR (code_employment = F AND Age ≥ 55)"

or

"code_contribution ≠ V OR (code_employment = F AND Age ≥ 55)"

Applying coverage type MCDC produces the following four test situations:

D1 A OR (B AND C)	1 valid input	0 error message
A: code_contribution ≠ V	1 1 0 (1)	0 1 0 (3)
B: code_employment = F	0 1 1 (2)	0 0 1 (4)
C: age ≥ 55	~~0 1 1~~	~~0 1 0~~

2- Creating logical test cases

The test situations from step 1 are one-on-one the logical test cases.

Example

Working out the four test situations from our example immediately gives us the four logical test cases:

Test cases/ Test situations	**D1-1**	**D1-2**	**D1-3**	**D1-4**
Customer	in NL	in NL	not in NL	in NL
Postal code	not in NL	in NL	not in NL	in NL
Country code	31	≠ 31	31	31
Expected result	Error message	Error message	OK	OK

3- Creating physical test cases

No remarks.

4- Establishing the starting point

No remarks.

3.7.6 Decision Table Test (DTT)

Characteristics

Approach	Coverage based - conditions (if applicable: data)

Quality characteristic / Test Variety	• Functionality
	  • Detailed functionality (functions that are being considered to be very important and/or complex calculations)
	  • Thorough coverage of conditions
	  • *Not*: combining functional paths

Coverage Type	• Decision points: Multiple Condition Coverage • If applicable: Data combinations
Test Basis	• Decision tables, pseudo-code, a process description or other (functional) descriptions that contain conditions

Description

The decision table test is a thorough technique for the testing of detail functionality. The required test basis contains conditions or decision tables. The type and structure of this test basis is of minor importance to the application of the decision table test technique.

The decision table test is aimed at the thorough coverage of the conditions and not at combining functional paths. The coverage type used here is:

- Decision points: multiple condition coverage

Variations on the decision table test can be created by applying other coverage types:

- Decision points: condition coverage, decision coverage or condition/decision coverage
  With these, *less test intensity* is achieved.
- Boundary value analysis
  With this, a condition can be tested with a more thorough test intensity.

This technique will mainly be chosen for testing functions and/or complex calculations considered to be very important.

Points of focus in the steps

In this section, the decision table test is explained step by step, taking the generic steps as a starting point (see section 3.2.2. The generic test design steps). An example is used at every step to show how this technique works.

The test basis consists of decision tables, pseudo-code, a process description or other (functional) descriptions, in which conditions occur. The conditions and the results are put into a decision table. The general occurrence of a decision table is shown in the table below.

Identification table				
Test situations	**1**	**2**	**..**	**n**
Condition 1	0	0	..	1
Condition 2	0	~	..	0

Identification table				
Test situations	**1**	**2**	**..**	**n**
Condition ..	0	..	~	0
Condition n	0	1	..	0
Result 1	X	..	..	..
Result ..	..	..	..	..
Result n	..	..	..	..

Each column of the decision table forms a test situation. The part above the double line forms the situation description and the part below the line reflects the consequences, or the results.

The conditions can either have the values of "0" or "1" (see section 3.4.2). The value "1" means that the condition is true; the value "0" means that the condition is false. The value "~" can also be allocated. This means that the value of the condition is not important. Below the double line, the cells contain an "X", or are empty. Where there is an "X", the relevant result occurs in that test situation; if a cell is empty, the relevant result does not occur in that test situation. Several results are possible per test situation. "Not possible" indicates that the test situation is not physically executable, for example because certain values of conditions exclude each other.

Example

When ordering coffee capsules via the Internet, the shipping costs are calculated. These consist of the standard shipping costs, plus a long-distance supplement. The text below shows the associated process description.

Shipping costs calculation:
Calculation of standard shipping costs
If 200 or more capsules are ordered and if the form of payment is "direct debit", then no shipping cost is applied. If fewer than 200 capsules are ordered, or if the form of payment is other than "direct debit", then a shipping cost of €10 is applied.

Calculation of long-distance supplement
If the delivery address for the capsules is within a radius of 50 km of Utrecht, no long-distance supplement is applied. If the delivery address is 50 km or more from Utrecht, but still in the Netherlands, then a long-distance supplement of €5 is applied. If the delivery address is outside of the Netherlands, then a long-distance supplement of €15 is applied. (The highest sum is applied.)

Below, each step is set out showing how the decision table test is applied to this process:

1- Identifying test situations

To fill in the table, in step 1 "Identifying test situations" the following activities are carried out:

- Find conditions in the test basis
- Create a conditions list
- Find results in the test basis and add these to the conditions list
- Fill in the decision table.

The activities are explained below:

Revealing the conditions involves quite some detective work. Often, a condition in the test basis is preceded by words such as "as long as", "if" and "then" and can be searched for by looking for these words.

Example

The tester has underlined the conditions in the process description.

Shipping costs calculation:
Calculation of standard shipping costs
If 200 or more capsules are ordered *and* if the form of payment is "direct debit", then no standard shipping costs are applied. If fewer than 200 capsules are ordered *or* if the form of payment is other than "direct debit", then a standard shipping cost of €10 is applied.

Calculation of long-distance supplement
If the delivery address for the capsules is within a radius of 50 km from Utrecht, then no long-distance cost is applied. If the delivery address is 50 km or more from Utrecht, but still in the Netherlands, then a long-distance supplement of €5 is applied. If the delivery address is outside of the Netherlands, then a long-distance supplement of €15 is applied.

Subsequently, a conditions list is created. If the test basis is a decision table, the conditions can often be copied one for one. In creating the list, the following rules are applied. These rules are created in order to keep the decision tables clear and intelligible:

- A condition is singular (meaning: without "AND" or "OR" constructions)
- A condition is formulated positively (in order to avoid "not not" combinations)
- Try to keep the number of conditions per table to five or lower (that is maximum $2^5 = 32$ test columns). If there are more conditions, split the table into several tables.

Example

Concerning the shipping costs calculation, the tester arrives at the following conditions list:

Calculation of standard shipping costs
C1 order ≥ 200 capsules
C2 form of payment = "direct debit"

Calculation of long-distance supplement
C3 distance < 50 km from Utrecht
C4 country = The Netherlands

Creating a conditions list may require some interpretation of the description. There often appears to be more conditions necessary in order to reach a particular situation. In that case, investigate whether there is indeed a supplementary condition or if a particular situation can be realized by one or more of the recognized conditions being false.

When the conditions list is known, the results are added to it. The tracing of the results also involves some detective work. A result is often preceded in the test basis by words such as "then" and "else".

Example

The tester has underlined the results in the process description.

Shipping costs calculation:
Calculation of standard shipping costs

If 200 or more capsules are ordered and if the form of payment is "direct debit", then no standard shipping costs are applied. If fewer than 200 capsules are ordered or if the form of payment is other than "direct debit", then a standard shipping cost of €10 is applied.

Calculation of long-distance supplement
If the delivery address for the capsules is within a radius of 50 km from Utrecht, then no long-distance supplement is applied. If the delivery address is 50 km or more from Utrecht, but still in the Netherlands, then a long-distance supplement of €5 is applied. If the delivery address is outside of the Netherlands, then a long-distance supplement of €15 is applied.

The tester adds the results to the conditions list:

Calculation of standard shipping costs
C1 Order ≥ 200 capsules
C2 Form of payment = "Direct debit"
R1 Standard shipping costs := 0
R2 Standard shipping costs := 10

Calculation of long-distance supplement
C3 Distance < 50 km from Utrecht
C4 Country = The Netherlands
R3 Long-distance supplement := 0
R4 Long-distance supplement := 5
R5 Long-distance supplement := 15

Now that both conditions and results are known, the decision table is filled in using coverage type decision points: multiple condition coverage. This means: all possible combinations of the values (0 or 1) of the separate conditions.

Example

Since the total number of conditions amounts to four, the tester has decided to include these in one table[3]. The conditions list and the filling in of the tables according to multiple condition coverage produces the table below with test situations for the shipping costs example:

Shipping costs calculation (test situations)																
Std. shipping costs / Long-distance supplement	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
C1 Order ≥ 200 capsules	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
C2 Form of payment = "direct debit"	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
C3 Distance < 50 km from Utrecht	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0
C4 Country = The Netherlands	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
R1 Std. shipping cost := 0									X	X	X	X				
R2 Std. shipping cost := 10	X	X	X	X	X	X	X	X					X	X	X	X
R3 Long-distance supplement := 0			X	X	X	X					X	X	X	X		
R4 Long-distance supplement := 5		X					X			X					X	
R5 Long-distance supplement :=	X			X	X			X	X			X	X			X

[3] This is irrelevant to the final number of combinations. Consider: in the shipping costs example, one table is created with four conditions in total. This leads to one table with maximum $2^4 = 16$ combinations. In the example, the table could be split into two tables ("Calculation of standard shipping cost" and "Calculation of long-distance supplement"). Both tables would then consist of $2^2 = 4$ test columns. Those, in combination with each other, would give 4 x 4 =16 combinations. Since splitting or nor splitting the table makes no difference to the final result, it is advisable to split tables with more than five conditions into several tables, since this makes the individual tables clear and intelligible.

Shipping costs calculation (test situations)																
Std. shipping costs / Long-distance supplement	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
15																

Reading a decision table is often considered to be difficult. Test situation 7, for example, should be read as follows:
The customer has ordered fewer than 200 capsules AND has selected the "direct debit" payment form AND the delivery address is 50 km or more from Utrecht AND the delivery address is in the Netherlands. The shipping costs amount to €10 standard cost plus €5 long-distance supplement, equals €15.

Filling in a table with ones and zeros can be done in many ways. The manner of doing so in the above "Shipping costs calculation" table simplifies the creation of physical test cases (see "In more detail" below for explanation).

In more detail

Note the clever way of filling in the "Shipping costs calculation" decision table. This causes only one condition to change in value per column (referred to in the literature under the name of: "Gray-code"). This is helpful for the creation of the physical test cases: copy and paste and change one value. For the filling in, we begin at the bottom row of conditions with one 0 followed by, consecutively, two times 1, two times 0 and so on until the last, which is given the value 0. In the row second from the bottom, we begin with two times 0 followed by, consecutively, four times 1, four times 0, and so on until the last two, which are given the value 0. We continue like this with the whole table; in every row the zero and one sets are twice as long as in the previous row.

In more detail

Instead of filling in the decision table according to coverage type multiple condition coverage, a (more elementary) variant could be chosen at the stage when the strategy is being determined. This technique is applied both to the conditions and to the results. As an example, the "shipping costs calculation" table has been filled in according to condition/decision coverage:

Shipping costs calculation (test situations)			
Std. shipping costs / long-distance supplement	**1**	**2**	**11**
C1 Order ≥ 200 capsules	0	0	1
C2 Form of payment = "direct debit"	0	0	1
C3 Distance < 50 km from Utrecht	0	0	1
C4 Country = The Netherlands	0	1	1
R1 St. shipping costs := 0			X
R2 St. Shipping cost := 10	X	X	
R3 Long-distance supplement := 0			X
R4 Long-distance supplement := 5		X	
R5 Long-distance supplement := 15	X		

With the three columns, all the possible outcomes of each condition and of each result are covered at least once. The columns 1 and 11 cover all the conditions, and column 2 is necessary to cover result 4 as well.

Several combinations of columns are possible that meet with the condition/decision coverage (e.g. 3, 9, 10 and 2, 11, 16).

In more detail

Besides conditions that can only have the values of true or false, parameters also exist with more than 2 possible values (i.e.: equivalence classes). Testing all combinations in that case is a coverage type that belongs to the group Data – Data combinations (instead of Conditions – Decision Points, or – like in the example below – in both groups):

- Add as many columns as there are possible equivalence classes, whereas the content of the other rows of conditions does not change. Suppose that in the example there is a choice of three forms of payment: direct debit, giro transfer and cash. Then, as an example, the 'old' test situation 1 would lead to the following 3 'new' test situations in this approach:

Std. shipping costs / long-distance supplement	**1**	**2**	**3**
C1 Order ≥ 200 capsules	0	0	0
C2 Form of payment	"direct debit"	"giro tr."	"cash"
C3 Distance < 50 km from Utrecht	0	0	0
C4 Country = The Netherlands	0	0	0
R2 St. Shipping cost := 10	X	X	X
R5 Long-distance supplement := 15	X	X	X

2- Creating logical test cases

The test situations (columns) in step 1 now constitute the logical test cases. However, a logical test case must not contain 'mutually exclusive conditions', since that would make the test case inconsistent in itself, and therefore unexecutable. In the step from test situations to logical test cases, any unexecutable test cases should be traced. These test cases are marked "Not possible" in the table.

Example

In the shipping costs example, the conditions C3 and *not*-C4 exclude each other. There are no foreign locations within a radius of 50 km from Utrecht. Therefore, 4 of the 16 logical test cases are unexecutable, see the table below:

Shipping costs calculation (logical test cases)																
Std. shipping costs / long-distance supplement	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
C1 Order ≥ 200 capsules	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
C2 Form of payment = "direct debit"	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
C3 Distance < 50 km from Utrecht	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0
C4 Country = The Netherlands	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
R1 Std. shipping cost := 0									X	X	X					
R2 St. Shipping cost := 10	X	X	X			X	X	X						X	X	X
R3 Long-distance supplement := 0			X			X					X			X		
R4 Long-distance supplement := 5		X					X			X					X	
R5 Long-distance supplement := 15	X							X	X							X
Not possible				X	X							X	X			

Check for yourself that the logical test cases 4, 5, 12 and 13 are not executable.

For the logical test cases with the result of "Not possible", no physical test case exists. Ideally, these columns should not be removed, so that no misunderstanding can arise as to why no physical test case is present for a particular logical test case.

3- Creating physical test cases

With a physical test case, all the data that play a part in the conditions are translated into concrete terms. To this end, the table of logical test cases can simply be adapted for making the test cases physical. In the table of physical test cases, each numbered column describes a physical test case and the last row(s) contain(s) the predicted result(s). For the entries:

- The "0" or the "1" is replaced in the table with a physical value
- The physical value is entered in the place of an "X".

A point of focus with the first bullet is that in the table of physical test cases, no conditions remain, only data. A particular data attribute can occur in several conditions. In logical test cases, they occur in several rows, while that particular data naturally only occurs once in the table for physical test cases. Besides this, it is possible that additional refinements are required. For example, by putting derived data into concrete terms (see example below).

Example

For the creation of physical test cases, the "Place of delivery" has to be derived from "Distance from Utrecht". This delivers the table 14.8 (as an example, 8 of the 12 logical test cases are shown):

Shipping costs calculation (physical test cases)								
	1	**2**	**6**	**7**	**9**	**10**	**11**	**16**
Number of capsules	199	199	199	199	200	200	200	200
Form of payment	cash	cash	dir.dbt.	dir.dbt.	dir.dbt.	dir.dbt.	dir.dbt.	cash
Distance from Utrecht	178	182	10	182	178	182	10	178
Place of delivery	Brussel	Heerlen	Zeist	Heerlen	Brussel	Heerlen	Zeist	Brussel
Country	B	NL	NL	NL	B	NL	NL	B
St. shipping cost	10	10	10	10	0	0	0	10
Long-distance suppl.	15	5	0	5	15	5	0	15
Total shipping costs	25	15	10	15	15	5	0	25

The logical test cases 4, 5, 12 and 13 are not executable and can therefore not be made physical.

In more detail

The decision table test can be made even more thorough by the applying boundary value analysis. This option is included as an extra condition in the creation of the logical test cases. In the example, this condition could be included in respect of the number of capsules and the distance. The requirement then is that for the "number of capsules" at least the values of 199, 200 and 201 should occur (e.g. in columns 9, 10 and 11) and for "distance" at least the values of 49, 50 and 51 (e.g. in columns 1, 2 and 7).. The number of test cases does not change with this approach.

Another possibility is to include a separate column for each value. This is a thorough method that tests all the combinations, but it is labor-intensive. The number of test cases increases with this approach.

4- Establishing the starting point

No remarks.

Example

The customer's details that are relevant to the placing of an order should be present in the information system.

3.7.7 Elementary Comparison Test (ECT)

Characteristics

Approach	Coverage based – conditions

Quality characteristic / Test Variety	• Functionality
	  • Detailed functionality
	  • Thorough coverage of decision points
	  • *Not*: combining functional paths

Coverage Type	• Decision points: Modified Condition / Decision Coverage
Test Basis	• Decision tables, pseudo-code, a process description or other (functional) descriptions that contain conditions and decision points

Description

The elementary comparison test (ECT) is a thorough technique for the detailed testing of the functionality. The necessary test basis is pseudo-code or a comparable specification in which the decision points and functional paths are worked out in detail and structurally. The ECT aims at thorough coverage of the decision points and not at the combining of functional paths. The coverage type used here is:

- Decision points: Modified Condition / Decision coverage

Variations on the ECT can be created by the application of the following coverage types:

- Decision points: Multiple Condition coverage
  With this, the possibilities within the decision points (specifically selected, if necessary) can be tested more thorough.
- Decision points: Condition coverage up to and including Condition / Decision coverage
  With this, the possibilities within the decision points (specifically selected, if necessary) can be tested less thorough.
- Boundary value analysis
  With this, the possibilities within the decision points (specifically selected, if necessary) can be tested more thorough.
- Pairwise testing
  With this, the testing of possible combinations of functional paths is added.

This technique will mainly be chosen for testing functions and/or complex calculations considered to be very important.

Points of focus in the steps

In this section, the decision table test is explained step by step, taking the generic steps as a starting point (see section 3.2.2. The generic test design steps). An example is used at every step to show how this technique works.

Example

In this example, we take a function (task) in which the data referring to the car owner are entered in a screen and subsequently, upon request, a calculation is made of the premium that the car owner should pay for his vehicle insurance. Depending on a number of variables, the level of the premium is established. The pseudo-code below gives a detailed functional description of this:

```
IF                          age < 18 years OR driving licence suspended
THEN                        error message
ELSE        IF              age < 25 years AND years holding driving licence < 3
            THEN            premium := 1500
            ELSE            premium := 800
            ENDIF
            IF              car age < 2 OR (car age ≥ 5
                            AND damage in last 3 years ≥ 2500)
                            OR age ≥ 70
            THEN            increase premium by 500
            EINDIF
EINDIF
```

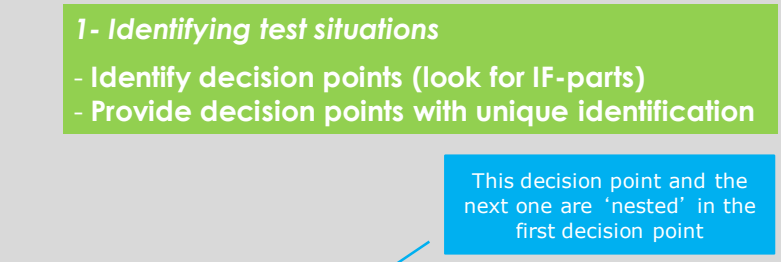
It is set out per step below how the elementary comparison test is applied to this function.

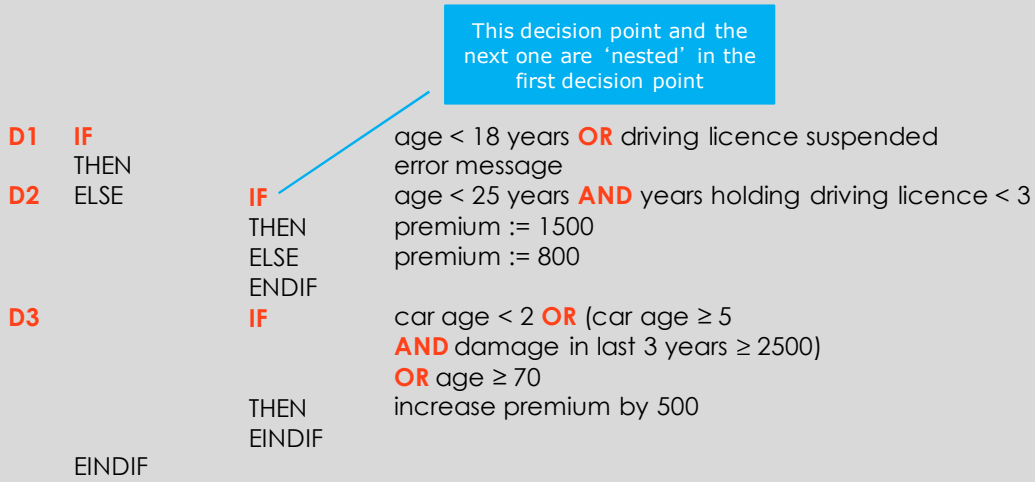
1- Identifying test situations

The test basis consists of pseudo-code or a comparable formal function description which can be copied directly in this step. If not, an extra activity should be carried out in order to convert the existing specifications into pseudo-code.
The decision points in the pseudo-code are provided with unique identification. It is usual to use the codes D1, D2, etc. for this (or D01, D02, etc. if there are many decision points).

Example

There are three decision points, which are identified below:





D2 and D3 are 'nested' decision points. Which means that getting to these decision points depends on the result of a previous decision point. In this case, D2 and D3 are only reached if the outcome of D1 is false.

Per decision point, the coverage type Decision points: MCDC (modified condition/decision coverage) is applied. The resulting test situations are numbered. The combination of this number and the decision point provides a unique identification of the test situations (such as D1-1, D1-2, etc.). The numbering begins with the test situations from column "1" (True) and then from the column "0" (False).

For each decision point, the test situations are worked out in detail in a separate table. The rows of the table contain the data or parameters that occur in the conditions of the decision point. A column then indicates which requirements are set on each parameter for the relevant test situation.

Example

1- Identifying test situations

- Apply coverage type (e.g. MCDC) per decision point
- Uniquely identify test situations

D1 IF age < 18 years **OR** driving licence suspended
THEN error message
ELSE

Indicates the outcome (which may imply going to the next decision point)

D1 **A OR B**	**1** **error message**	**0** **(D2)**
A: age < 18	1 0 (1-1)	0 0 (1-3)
B: driving licence suspended	0 1 (1-2)	~~0 0~~

D2 IF age < 25 years **AND** years holding driving licence < 3
THEN premium := 1.500
ELSE premium := 800

D2 **A AND B**	**1** **Premium= 1500**	**0** **premium= 800**
A: age< 18	1 1 (2-1)	0 1 (2-2)
B: years holding driving licence < 3	~~1 1~~	1 0 (2-3)

D3 IF car age < 2 **OR** (car age ≥ 5
AND damage in last 3 years ≥ 2500)
OR age ≥ 70
THEN increase premium by 500

D3 **A OR (B AND C) OR D**	**1** **premium + 500**	**0**
A: car age < 2	1 0 1 0 (3-1)	0 0 1 0 (3-4)
B: car age ≥ 5	0 1 1 0 (3-2)	0 0 1 0
C: damage in last 3 years ≥ 2500	~~0 1 1 0~~	~~0 1 0 0~~ (3-5)
D: age ≥ 70	0 1 0 1 (3-3)	~~0 1 0 0~~

NB! In D3, the combination "A = true and B = true" gives a logical contradiction and therefore may not occur in the test situations: Car age should be simultaneously lower than 2 and higher than, or equal to, 5. This contradiction would otherwise show up when the test cases are made physical.

Detailed working out of the derived test situations:

1- Identifying test situations
- Detailed elaboration of the derived test situations

D1 IF age < 18 years **OR** driving licence suspended
THEN error message
ELSE

D1 **A OR B**	**1** **error message**	**0** **(D2)**
A: age < 18	1 0 (1-1)	0 0 (1-3)
B: driving licence suspended	0 1 (1-2)	~~0 0~~

D1	**D1-1**	**D1-2**	**D1-3**
Age	**< 18**	**≥ 18**	**≥ 18**
Driving licence suspended	**N**	**Y**	**N**

1- Identifying test situations
- Detailed elaboration of the derived test situations

D1	**D1-1**	**D1-2**	**D1-3**
Age	< 18	≥ 18	≥ 18
Driving licence suspended	N	Y	N

D2	**D2-1**	**D2-2**	**D2-3**
Age	< 25	≥ 25	< 25
years holding driving licence	< 3	< 3	≥ 3

D3	**D3-1**	**D3-2**	**D3-3**	**D3-4**	**D3-5**
Car age	< 2	≥ 2	≥ 2	≥ 2	≥ 2
Car age	< 5	≥ 5	≥ 5	< 5	≥ 5
Damage in last 3 years	≥ 2500	≥ 2500	< 2500	≥ 2500	< 2500
Age	< 70	< 70	≥ 70	< 70	< 70

NB! The parameter "Age" occurs in the decision points D1, D2 and D3. This leads to the following mutually exclusive test situations: D2-1 with D3-3; D2-3 with D3-3.

2- Creating logical test cases

A test case runs through the functionality from start to end and will come across one or more decision points on its path. With each decision point, the test case will test one of the defined test situations.

The logical test cases are combined with the aid of a matrix.

In order to take account of the nesting of decision points, the columns "Value" and "Next" are added. These indicate for each test situation what the outcome of the decision is (directly obtainable from the tables in step 2) and to which subsequent decision point (or end process) this leads. This helps to prevent the tester from placing a cross at a test situation where the test case does not go. This can also be achieved by using a Graph (see below).

Graphic demonstration of test situations

For some testers, the creation of logical test cases is made easier with the aid of a graphic demonstration of the test situations – a *Graph*.
With this, each decision point and end point is represented by a circle and each test situation by a line that goes from one circle to another.

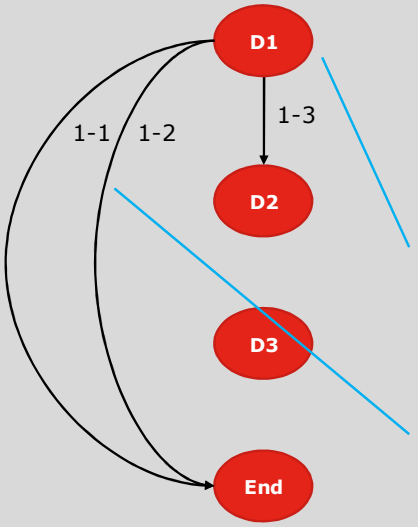
A logical test case runs through the graph from beginning to end, linking a chain of test situations. The graph also supplies insight into the minimum number of test cases necessary to cover all the test situations. This is determined by the maximum number of parallel lines in the graph.

Example

2- Creating logical test cases

- Graphic demonstration of relation between test situations

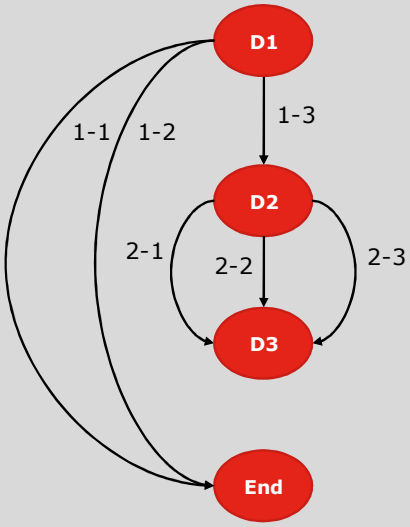
D1 **A OR B**	**1** **error message**	**0** **(D2)**
A: age < 18	1 0 (1-1)	0 0 (1-3)
B: driving licence suspended	0 1 (1-2)	~~0 0~~



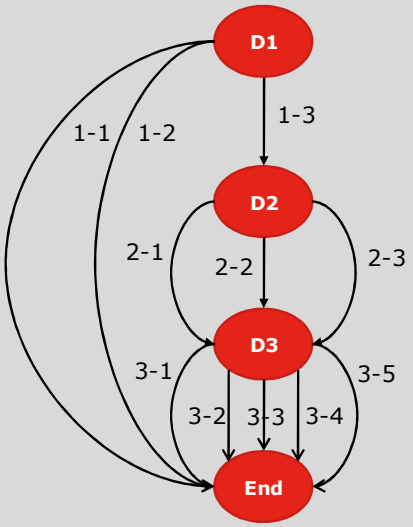
A circle for each decision point plus a circle for 'End"

Each test situation is drawn as a line, leading to its destination

D2 **A AND B**	**1** **premium= 1500**	**0** **premium=800**
A: age < 18	1 1 (2-1)	0 1 (2-2)
B: years holding driving licence	~~1 1~~	1 0 (2-3)



D3 A OR (B AND C) OR D	1 Premium + 500	0
A: car age < 2	1 0 1 0 (3-1)	0 0 1 0 (3-4)
B: car age ≥ 5	0 1 1 0 (3-2)	~~0 0 1 0~~
C: Damage in last 3 years ≥ 2500	~~0 1 1 0~~	0 1 0 0 (3-5)
D: age ≥ 70	0 1 0 1 (3-3)	~~0 1 0 0~~



Estimate the minimum number of test cases: the maximum number of parallel lines

Mutually exclusive test situations

Each test situation sets particular requirements on one or more parameters. If a parameter occurs in several decision points, it is possible that a test situation in one decision point sets requirements on that parameter that conflict with the requirements of a test situation in another decision point. For example, test situation D2.1 requires "Age < 25" and test situation D3.3 requires "Age ≥ 70". These test situations are mutually exclusive.

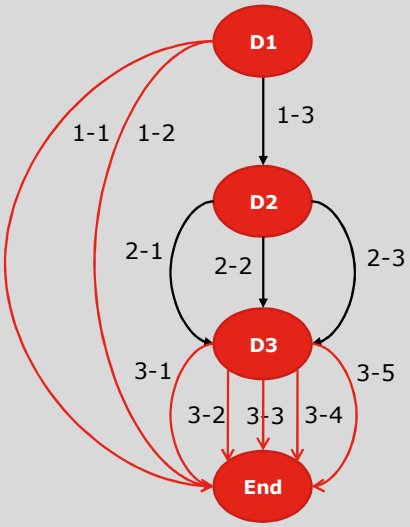
A logical test case can not contain "mutually exclusive test situations", for that makes the test case inconsistent and therefore unexecutable. Such a test case will be discovered automatically, as soon as the test case has to be made physical (see next step). The problem can then be simply resolved, by replacing one of the "mutually exclusive test situations" with a non-conflicting test situation. In this connection, it can be advantageous to first translate each logical test case into a physical test case in order to discover possible mutually exclusive test situations, before starting on the following logical test case.

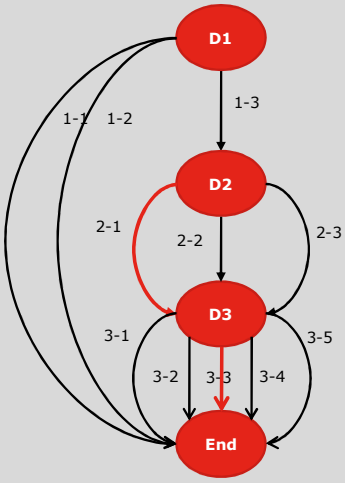
In order to prevent test cases from occurring, that contain mutually exclusive test situations, an extra analysis should be carried out in advance:

- Inventorize which parameters occur in several decision points, and (per parameter) which decision points they are
- Sum up the combinations of mutually exclusive test situations.

Example

2- Creating logical test cases
- Determine mutually exclusive test situations



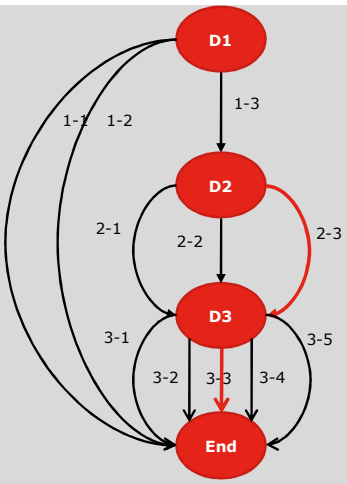


D1	D1-1	D1-2	D1-3
Age	< 18	≥ 18	≥ 18
Driving licence suspended	N	Y	N

D2	D2-1	D2-2	D2-3
Age	< 25	≥ 25	< 25
Years holding driving licence	< 3	< 3	≥ 3

D3	D3-1	D3-2	D3-3	D3-4	D3-5
Car age	< 2	≥ 2	≥ 2	≥ 2	≥ 2
Car age	< 5	≥ 5	≥ 5	< 5	≥ 5
Damage in last 3 years	≥ 2500	≥ 2500	< 2500	≥ 2500	< 2500
Age	< 70	< 70	≥ 70	< 70	< 70

Excluding: 2-1 and 3-3

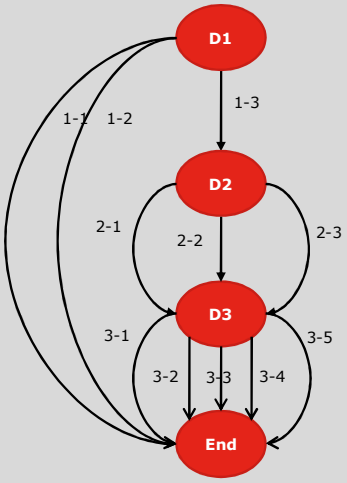


D1	D1-1	D1-2	D1-3
Age	< 18	≥ 18	≥ 18
Driving licence suspended	N	Y	N

D2	D2-1	D2-2	D2-3
Age	< 25	≥ 25	< 25
Years holding driving licence	< 3	< 3	≥ 3

D3	D3-1	D3-2	D3-3	D3-4	D3-5
Car age	< 2	≥ 2	≥ 2	≥ 2	≥ 2
Car age	< 5	≥ 5	≥ 5	< 5	≥ 5
Damage in last 3 years	≥ 2500	≥ 2500	< 2500	≥ 2500	< 2500
Age	< 70	< 70	≥ 70	< 70	< 70

Excluding: 2-3 and 3-3



D1	D1-1	D1-2	D1-3
Age	< 18	≥ 18	≥ 18
Driving licence suspended	N	Y	N

D2	D2-1	D2-2	D2-3
Age	< 25	≥ 25	< 25
Years holding driving licence	< 3	< 3	≥ 3

D3	D3-1	D3-2	D3-3	D3-4	D3-5
Car age	< 2	≥ 2	≥ 2	≥ 2	≥ 2
Car age	< 5	≥ 5	≥ 5	< 5	≥ 5
Damage in last 3 years	≥ 2500	≥ 2500	< 2500	≥ 2500	< 2500
Age	< 70	< 70	≥ 70	< 70	< 70

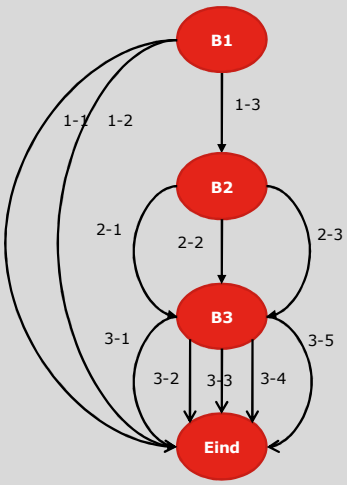
1-1 and 2-2 are not mutually excluding since 1-1 directly goes to the end and does not pass 2-2

Combining test situations to logical test cases

The final step in creating logical test cases is to combine the test situations to logical test cases, in such a way that every test situation is covered by *at least* one logical test cases. This is done with the aid of a matrix, taking into account the mutually exclusive test

situations. The rows contain the test situations and the columns contain the logical test cases. With each test case, it is indicated by one or more crosses which test situations should be tested by this test case. This matrix simultaneously serves as a check on the complete coverage of test situations.

Example



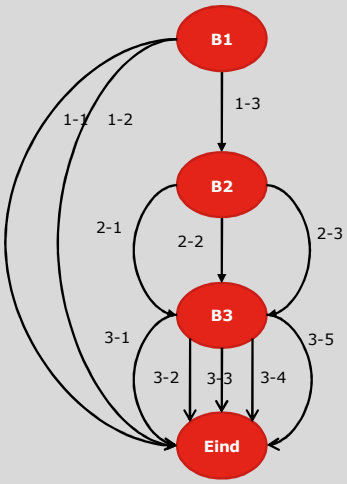
2- Creating logical test cases
- Combining test situations to logical test cases

Test situations	Value	Next
D1-1	1	End
D1-2	1	End
D1-3	0	D2
D2-1	1	D3
D2-2	0	D3
D2-3	0	D3
D3-1	1	End
D3-2	1	End
D3-3	1	End
D3-4	0	End
D3-5	0	End

With the aid of a matrix, taking the exclusions into account

The result of the test situation

Exclusions
D2-1 with D3-3
D2-3 with D3-3



2- Creating logical test cases
- Resulting matrix

Test situations	Value	Next	TC-1	TC-2	TC-3	TC-4	TC-5	TC-6	TC-7
D1-1	1	End	X						
D1-2	1	End		X					
D1-3	0	D2			X	X	X	X	X
D2-1	1	D3			X				
D2-2	0	D3					X		X
D2-3	0	D3				X		X	
D3-1	1	End			X				
D3-2	1	End				X			
D3-3	1	End					X		
D3-4	0	End						X	
D3-5	0	End							X

Exclusions
D2-1 with D3-3
D2-3 with D3-3

If necessary, every logical test case can be elaborated further.

Example

2- Creating logical test cases
- Further elaboration

Test situations	TC-4
D1-1	
D1-2	
D1-3	X
D2-1	X
D2-2	
D2-3	
D3-1	X
D3-2	
D3-3	
D3-4	
D3-5	

Test case	TC-4
Test situations	D1-3 D2-1 D3-1
Age	≥ 18 and < 25
Licence suspended	N
Years licence	< 3
Car age	< 5
Damage 3 yrs	≥ 2500
Result:	
Error message	-
Premium	2000

D1	**D1-1**	**D1-2**	**D1-3**
Age	< 18	≥ 18	≥ 18
Licence suspended	N	Y	N

D2	**D2-1**	**D2-2**	**D2-3**
Age	< 25	≥ 25	< 25
Years licence	< 3	< 3	≥ 3

D3	**D3-1**	**D3-2**	**D3-3**	**D3-4**	**D3-5**
Car age	< 2	≥ 2	≥ 2	≥ 2	≥ 2
Car age	< 5	≥ 5	≥ 5	< 5	≥ 5
Damage 3 yrs	≥ 2500	≥ 2500	< 2500	≥ 2500	< 2500
Age	< 70	< 70	≥ 70	< 70	< 70

Since these are logical test cases, the values are still at a logical level. Please note that Age occurs in D1, D2 *and* D3. So age has to comply with the values in D1-3, D2-1 *and* D3-1, and therefore must be ≥ 18 and <25.

3- Creating physical test cases

With a physical test case, all the parameters (data) have to be given concrete substance, so that the relevant test situations are covered by this.

Physical test cases can be handily described with the aid of a matrix that is built up as follows:

- Each column describes a physical test case.
- The first row indicates per test case which test situations should be covered.
- Thereafter, there is a row for each parameter of which the test case consists.
- Finally, one or more rows are added with which the predicted result is described in concrete terms.

4- Establishing the starting point

No remarks.

5- Creating the test script

No remarks.

3.7.8 Data Cycle Test (DCyT)

Characteristics

Approach	Coverage based – data and conditions
Quality characteristic / Test Variety	• Overarching functionality • Suitability • Connectivity
Coverage Type	• CRUD (for covering the life cycle of data) • Decision points: Decision Coverage (for covering integrity rules)
Test Basis	• CRUD matrix • Functional description

Description

The data cycle test (DCyT) is a technique for testing whether the data are being used and processed consistently by various functions from within different subsystems or even different systems. The technique is ideally suited to the testing of overall functionality, suitability and connectivity.

The primary aim of the data cycle test is not to trace functional defects in individual functions, but to find integration defects. The test focuses on the link between various functions and the way in which they deal with communal data. The DCyT is most effective if the functionality of the individual functions has already been sufficiently tested. That is also an important reason why this test is usually applied in the later phases of acceptance testing.

The most important test basis is the CRUD matrix (see section 3.5.6) and a description of the applicable integrity rules. The latter describe the preconditions under which certain processes are or are not permitted, such as, for example, "Entity X may only be changed if the linked entity Y is removed from it". Besides this, functional specifications or detailed domain expertise is necessary in order to be able to predict the result of each test case.

The coverage types used are:
- CRUD, for coverage of the life cycle of the data
- Decision coverage, for coverage of the integrity rules.

The test intensity of the test can be increased by applying e.g.:
- a more extended variant of CRUD
- Modified condition/decision coverage or multiple condition coverage of the integrity rules.

Points of focus in the steps

In this section, the data cycle test is explained step by step. In this, the generic steps (see section 3.3.2) are taken as a starting point. An example is also set out that demonstrates, up to and including the designing of the logical test cases, how this technique works.

1- Identifying test situations

The test situations are created from the coverage of the CRUD and from the integrity rules. Both will be further explained here.

Test situations in connection with CRUD
The following activities should be carried out:

- Determine the entities of which the life cycle is to be tested.
  Usually, this concerns all the entities that are used by the system or subsystem (created, changed, read or removed). If there are too many entities, a cohesive subset of entities may be selected
- Determine the functions that make use of these entities.
  Here, too, the scope of the test should be determined: all the functions of the system under test, a cohesive subset of this, functions from other systems that are linked to the system under test
- Fill in the CRUD matrix (see section 3.5.6).
  If the CRUD matrix is delivered as a test basis, the relevant part should be selected from this, based on the previous two activities. If it was not possible to get the CRUD matrix delivered as a test basis, the test team may decide to create this themselves, based on the functional specifications. This is obviously undesirable, but is a last resort
- Each process (C, R, U or D) that occurs in the CRUD matrix is a separate test situation that has to be tested.

Test situations in connection with integrity rules
The following activities should be carried out:

- Gather the integrity rules on the selected entities.
  These are the rules that define under which conditions the processing of the entities is valid or not. Integrity rules are usually specified within the functional specifications, database models or in separate business rules.
- Apply decision coverage. That means that for each integrity rule, two test situations are derived:
  - Invalid
    The integrity rule is not met. The process is invalid and should result in correct error handling.
  - Valid
    The integrity rule is met. The process is valid and should be executed.

In more detail

Integrity rules should not be confused with semantic rules, which define the conditions under which the *value* of the data themselves is valid or not. For example:

- The rule "When creating an order, the value of quantity should not be below the boundary that is set in product" – is a semantic rule
- The rule "The creation of an invoice is only permitted if the order concerned has already been approved" – is an integrity rule.

Therefore, the integrity rule determines whether the function is permitted in the first place. Thereafter, the semantic rules determine whether the input data offered to that function are valid.

Example

The data cycle test is applied to a subsystem that invoices orders and processes payments. The relevant part of the CRUD matrix is shown in the table below.

	Item	Payment agreement	Invoice	Ledger
Item management	C, R, U, D	-	-	…
Payment agreement management	-	C, R, U, D	R	…
Ledger management	-	-	R	C, R, U, D
Invoice creation	R	R	C	U
Cash payment	-	-	C, U, D	U
Bank transfer	-	-	U, D	U
…	…	…	…	…

For this part of the CRUD matrix, there is one relevant integrity rule: A payment agreement may not be removed as long as there is an outstanding invoice with the relevant payment agreement.

This leads to two test situations:

IR1-1: Delete (D) payment agreement, while an invoice is outstanding with the relevant payment agreement

IR1-2: Delete (D) payment agreement, without there being an outstanding invoice with the with the relevant payment agreement

A brief overview notation for this type of test situation is, for example:

Test situation	Process	Entity	Condition	Valid Y/N
IR1-1	D	Payment agreement	outstanding invoice	N
IR1-2	D	Payment agreement	no outstanding invoice	Y

The initials "IR" here stand for "Integrity rule".

2- Creating logical test cases

Create 1 or more logical test cases in such a way that:

- Each entity goes through a full life cycle (beginning with 'C' and ending with 'D')
- All the test situations from the CRUD matrix (every C, R, U and D) are covered
- All the test situations of the relevant integrity rules are covered.

See also section 3.5.6 and section 3.5.7.

A test case thus describes a complete scenario consisting of several actions, each of which perform a process on a particular entity.

Example

In the two tables below the logical test cases for the entities "Item" and "Payment agreement" are set out, to illustrate the principle.

The table describe at each row which function should be used, which process (CRUD) on the relevant entity is covered by this and a brief explanation with additional information on the action to be performed.

LTC-01: "Item"

Function	CRUD	Action / Notes
Item management	C	Create new item ITM
Item management	R	Check ITM
Create invoice	R	Create invoice INV
Ledger management	-	Check INV
Item management	U	Change ITM
Item management	R	Check ITM
Ledger management	-	Check INV
Item management	D	Remove ITM
Item management	R	Check ITM

LTC-02: "Payment agreement"

Function	CRUD	Action / Notes
Payment agreement mgt.	C	Create new payment agreement PAG
Payment agreement mgt.	R	Check PAG
Payment agreement mgt.	U	Change PAG
Payment agreement mgt.	R	Check PAG
Create invoice	R	Create invoice INV
Ledger management	-	Check INV
Payment agreement mgt.	D	IR1
Payment agreement mgt.	R	Check PAG
Cash payment	-	Full payment of INV
Payment agreement mgt.	D	IR1
Payment agreement mgt.	R	Check that PAG

A "-" in the column "CRUD" means that the relevant function is required in order to carry out a certain action, but that this does not perform any processing on the tested entity. For example:

With LTC01, "Ledger management" is used to be able to check that the correct item appears on the invoice, but does not perform any processing itself on "Item".

With LTC02, "Cash payment" is used to close invoice INV 02 so that integrity rule IR1-2 is complied with, but does not perform any processing itself on "Payment agreement".

3- Creating physical test cases

In the translation of logical test cases to physical test cases, the following details are added:

- (Optional) Exactly how the relevant function is activated. This is usually clear enough, but sometimes it requires a less obvious sequence of actions.
- The data to be entered with that function. If the logical test case indicates that a certain entity has to be changed, then the physical test case should indicate unequivocally which attribute is changed into which value.
- A concrete description with each predicted result of what has to be checked concerning a particular entity.
- Extra actions that are necessary to facilitate subsequent actions in the test case. E.g., the changing of the system date or the execution of a particular batch process in order to give the system a certain required status.

4- Establishing the starting point

The DCyT typically operates at overall system level, possibly across several systems. That means an extensive starting point has to be prepared that is complete and consistent across all the systems. The following, in particular, should be organized:

- All the necessary databases for all the systems involved, in which all the data is consistent
- A configuration (possibly a network) in which all the necessary systems are connected and in which all the necessary users are defined with the necessary access rights.

Such a starting point approximates the production situation and is complicated to put together. Ideally, an existing real-life test environment is used. See also section 4.3, under "Specification Phase": "Defining central starting point(s)".

In particular, attention should be paid to the data in the starting point that are only valid for a limited period of time. At the start of each test execution, it should be checked whether these time-dependent data are still valid and whether, on the basis of this, changes should be made in the starting point.

3.7.9 Real-Life Test (RLT)

Characteristics

Approach	Coverage based – Appearance
Quality characteristic / Test Variety	• Usability • Connectivity • Continuity • Performance
Coverage Type	• Operational profiles: sequence of transactions • Load profiles: numbers of users and/or transactions
Test Basis	• 'Profiles' = description of realistic usage

Description

With the real-life test (RLT), it is not the intention to test the system behaviour in separate situations, but to simulate the realistic usage of the system in a statistically responsible way. This test mainly focuses on characteristics, such as effectivity, connectivity, continuity and performance of the system under test. Many defects that are found with a real-life test are connected with a system's use of resources:

- Crashing of transactions following lengthy use
- Crashing of transactions that are carried out in a particular sequence
- Inadequate response times and speed of processing
- Insufficient memory or storage space available
- Insufficient capacity of peripherals and data-communication network.

To be able to test whether a system can handle realistic usage of it, that usage should be somehow specified. This also serves as a test basis and, in this context, is often referred to as the profile. The two most common types are:

- Operational profile
  Simulation of the realistic usage of the system, by carrying out a *sequence of transactions*, which is compiled in a statistically responsible way (see section 3.6.4).
- Load profile
  Simulation of a realistic loading of the system in terms of *numbers of* users and/or transactions (see section 3.6.3).

In practice, in a real-life test a mix of these profiles is often used. A particular loading of the system is simulated by carrying out realistic scenarios.

A profile is used in the setting out of one or more test goals of the real-life test. Examples of test goals are:

- Testing with normal or average usage
  The aim here is to examine whether the available system resources are sufficient for the usual circumstances. This often involves a test with an average number of users who carry out interactive work, run overviews and carry out a number of small batch functionalities.
- Testing with intensive usage
  The aim here is to examine whether there are sufficient system resources for even the most stressful, but realistic, circumstances. This often involves a test with a maximum

number of users who carry out interactive work (peak loading) or a test in which certain transactions are carried out often and at length.

- Measuring the breaking point (stress testing)
  The aim here is to examine what the maximum load is under which the system will still perform to an acceptable level. This often involves a test with an increasing number of (simulated) users.
- Testing daily batches
  The aim here is to examine whether the available system resources are sufficient for the combination of a normal number of interactive users with the simultaneous execution of relatively demanding batch jobs.
- Testing nightly batches
  The aim here is to examine whether both the available system resources and the available time are sufficient for the (nightly / weekend) execution of big batch jobs.

Execution of the real-life test is usually more complicated than that of other tests. In an environment in which the number of end users is not too great, you can have everyone work overtime for a weekend and carry out a previously established test scenario. However, use is increasingly being made of tools that simulate a realistic load in various ways. These are tools that simulate, for example, the number of users through the creation of virtual users, or tools that simulate a particular loading of the back-end of the system by offering transactions directly via the database management interface (hence without the use of the front-end or network).

It should be clearly determined in advance what and how measuring is to be done during the real-life test. Sometimes the measuring in itself also puts demands on the system, which can lead to distortion of the results. On the other hand, sufficient data is required to be able to carry out a satisfactory analysis in retrospect.

It can sometimes be difficult to assess the results of a real-life test. Occasionally, tests are not reproducible, because defects are often found that are caused by insufficient memory, lengthy use, etc. These kinds of defects (e.g. memory leaks) are difficult to reproduce, because there are almost always outside influences at play, which are impossible, or almost impossible, to control, such as the memory management of the operating system. In tracing the causes of any defects, logging and monitoring facilities could be used.

Points of focus in the steps

For the real-life test, the creation or establishing of correct profiles is the most important step. This takes place within step 1 "Identifying test situations". The exact content of the test cases is less relevant than with most other test design techniques. The most important criterion is that reality regarding the size and frequency of use is approximated as closely as possible. This means that there is usually no point in creating logical test cases. The physical translation can often be made immediately to cover the required test situations.

1- Identifying test situations

The profiles can be seen as the situations to be tested. These indicate along general lines which types of actions (functions) are carried out over a particular period and the number of active users. This may be a number of daily cycles, e.g. a minimum, average and maximum cycle. A daily cycle consists, for example, of logging on, intensive use, lighter use during the lunch break, intensive use, logging out, backup and daily batches. Besides these, there could also be comparable weekly, monthly and annual cycles and specific processes, such as backup and recovery. There are various ways of obtaining the

necessary information for the creation of an operational profile, load profile or a mix of these. Below are a number of them in random order:

- Derive the profile from the current system or release
- Copy an existing profile of a system with comparable functionality
- Copy an existing profile with comparable load of the system resources
- Log how often each function is used
- Measure the load of the system with the aid of specific tools (monitors)
- Interview users, in which the key question is: "Which transactions do you carry out, how often and when, or which transactions do you expect to carry out on the new system?"

An important consideration in creating a profile is the degree of detail. A more extensive and detailed profile will of course give a better reflection of reality, but will also lead to an increase in the testing effort for the specification and realization of test cases and the execution of the real-life test.

It should be ensured that in the profile all the system resources are used realistically. It is pointless to simulate significantly heavier usage than is usual in reality, as the result of such a test will tell us nothing. If, for example, the system is too slow under those circumstances, it does not mean that the system is unsatisfactory. If the system is not too slow, that only tells us that it is over-configured, but not by how much. Simulating significantly heavier usage is useful, of course, if the aim of the test is test is to determine the maximum load under which the system still performs acceptably.

Example

In an organization that processes bank transactions for various banks, 275,000 transactions are processed per hour with normal usage. These are divided into transaction types as follows (see the table):

Transaction type	Frequency(#/hr)	Relative frequency
Point of Sale transactions	150,000	0.55
Direct debits	90,000	0.33
ATM transactions	20,000	0.07
Credit transfers	15,000	0.05
Total	**275,000**	**1.0**

The test cases are realized to correspond with the tasks and the relative frequency. For the example, this means the following spread across the test cases: 55% point of sale transactions, 33% direct debits, 7% ATM transactions and 5% credit transfers.

Possible aims of the test are:
- What is the maximum number of transactions that can be processed per second?
- What is the average lead-time of a transaction under normal or intensive usage, and does this fall within the agreed limits?
- Is the system proof against lengthy uninterrupted use?

2- Creating logical test cases

Since the Real-Life Test is not about testing system usage in separate situations, logical test cases are usually not created. Creation of the physical test cases is started immediately.

3- Creating physical test cases

For the real-life test, the exact content of the physical test cases is less relevant than for most other test design techniques. The only criterion is that reality regarding size and

frequency of use is approximated as closely as possible. This sounds easier than it actually is. It has to be carefully considered *how* a particular usage or loading of the system can be realized or simulated. Additionally, test cases should be gathered or created for some tests, which have then to be carried out with the test execution. In contrast, in the execution of other tests the system has to be prepared with content *in advance*.

The creation of test cases can be done, for example, by physically setting out user scenarios, or, if an operational system already exists, by 'tapping' a representative test set.

For the testing of particular aspects of the system usage, the test cases can be realized by preparing a daily production (after processing) as real-life input. When using production data, bear in mind the privacy aspects. The devised user scenarios and actions that form part of the real-life test should also be distributed as realistically as possible among the users (testers) participating in the test.

The use of a tool, for example to simulate users or transactions, does not mean that user scenarios not need to be worked out. Even when a tool is used, these user scenarios form the basis of the test. In addition, the tool will have to be programmed or set so that it can carry out the user scenarios.

Example

In the example of the transaction processing, the following physical features have been given to the point of sale transactions (see the table):

Nr	Transaction type	Parameters*	Goal and expectation
1	Point of sale transactions	Let number of transactions in 90 minutes rise from 5 tr/sec to 450 tr/sec.	Determine breaking point (stress test). Expectation: ≥ 350 tr/sec.
2	Point of sale transactions	42 tr/sec (over 5 minutes)	Determine lead time of a transaction with normal use
3	Point of sale transactions	120 tr/sec (over 5 minutes)	Determine lead time of a transaction with intensive use
With test cases 2 and 3, besides the point of sale transactions, the fixed system load consists of 25 direct debits/sec, 5 cash withdrawals/sec and 4 credit transfers/sec. The transaction should be carried out to 95% < 5 sec, to 99% < 7 sec and to 100% < 10 sec.			
4	Point of sale transactions	42 tr/sec (over 7 days)	Test whether the system crashes with lengthy use

* The point of sale transactions amount to an average of €100.00 and are spread over 4 banks.

4- Establishing the starting point

As with most other tests, the creation of an appropriate starting point for a real-life test is often a challenge. However, with the real-life test, often there are additional points of focus:

- The test environment should be representative of the production situation
- Sizeable files are used
- Many users (testers) perform the testing
- A 'real' network should be available.

5- Creating the test script

No remarks.

Chapter 4 TMap NEXT info

4.1 What is testing?

While many definitions of the concept of testing exist, one way or another they all contain comparable aspects. Each of the definitions centers on the comparison of the test object against a standard (e.g. expectation, correct operation, requirement). With this, it is important to know exactly what you are going to test (the test object), against what you are going to compare it to (the test basis) and how you are going to test it (the test methods and techniques).

The International Standardisation Organization (ISO) and the International Electrotechnical Commission (IEC) apply the following definition [ISO/IEC, 1991]:

Definition
Technical operation that consists of the determination of one or more characteristics of a given product, process or service according to a specified procedure.

Testing supplies insight in the difference between the actual and the required status of an object. Where quality is roughly to be described as 'meeting the requirements and expectations', testing delivers information on the quality. It provides insight into, for example, the risks that are involved in accepting lesser quality. For that is the main aim of testing. Testing is one of the means of detection used within a quality control system. It is related to reviewing, simulation, inspection, auditing, desk-checking, walkthrough, etc. The various instruments of detection are spread across the groups of evaluation and testing:

- Evaluation : assessment of products without running software.
- Testing : assessment of products by means of running the software.

Put bluntly, the main aim of testing is to find defects: testing aims to bring to light the lack in quality, which reveals itself in defects. Put formally: it aims to establish the difference between the product and the previously set requirements. Put positively: it aims to create faith in the product.
The level of product quality bears a relationship to the risks that an organization takes when these products are put into operation. Therefore, in this book we define testing, according to TMap, as follows:

Definition
Testing is a process that provides insight into, and advice on, quality and the related risks.

Advice on the quality of what? Before an answer to this can be given, the concept of quality requires further explanation. What, in fact, is quality?

Definition
The totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs [ISO, 1994].

In aiming to convert 'implied needs' into 'stated needs' we soon discover the difficulty of subjecting the quality of an information system to discussion. The language for discussing

quality is lacking. However, since 1977, when McCall [McCall, 1977] came up with the proposal to divide the concept of quality into a number of different properties, the so-called quality characteristics, much progress has been made in this area.

Definition A quality characteristic describes a property of an information system.

A well-known set of quality characteristics was issued by the ISO and IEC [ISO 9126-1, 1999]. In addition, organizations often create their own variation of the above set. For TMap, a set of quality characteristics specifically suited to testing has been compiled, and these are listed and explained in chapter 10, "Quality characteristics and test types". This set is the one that is used within the framework of this book.

What, then, is the answer to the question: "Advice on the quality of what?"
Since, where quality is concerned, the issue is usually the correct operation of the software, testing can be summed up as being seen by many to mean: establishing that the software functions correctly. While this may be a good answer in certain cases, it should be realized that testing is more than that. Apart from the software, other test objects exist, the quality of which can be established. That which is tested, and upon which quality recommendations are subsequently given, is referred to as a test object.

Definition The test object is the information system (or part thereof) to be tested.

A test objects consists of hardware, system software, application software, organization, procedures, documentation or implementation. Advising on the quality of these can involve – apart from functionality – quality characteristics such as security, User-friendliness, performance, maintainability, portability and testability.

Pitfalls

In practice, it is by no means clear to everyone what testing is and what could or should be tested. Here are a few examples of what testing is *not*:

- Testing is not a matter of releasing or accepting something. Testing supplies advice on the quality. The decision as regards release is up to others (stakeholders), usually the commissioner of the test.
- Testing is not a post-development phase. It covers a series of activities that should be carried out in parallel to development.
- Testing is something other than the implementation of an information system. Test results are rather more inclined to hinder the implementation plans. And it is important to have these – often closely related - activities well accommodated organizationally.
- Testing is not intended initially to establish whether the *correct* functionality has been implemented, but to play an important part in establishing whether the *required* functionality has been implemented. While the test should of course not be discounted, the judgment of whether the right solution has been specified is another issue.
- Testing is not cheap. However, a good, timely executed test will have a positive influence on the development process and a qualitatively better system can be delivered, so that fewer disruptions will occur during production. Boehm demonstrated long ago that the reworking of defects costs increasing effort, time and money in proportion to the length of time between the first moment of their existence and the moment of their detection [Boehm, 1981]. See also "What does testing deliver?" in the next section.

- Testing is not training for operation and management. Because a test process generally lends itself very well to this purpose, this aspect is often too easily included as a secondary request. Solid agreements should see to it that both the test and the training will be qualitatively adequate. A budget and time should be made exclusively available for the training, and agreements made as regards priorities, since at certain times choices will have to be made.

It is the task of the test manager, among others, to see that these pitfalls are avoided and to make it clear to the client exactly what testing involves.

4.1.1 What is structured testing?

In practice, it seems that testing is still being carried out in an unstructured manner in many projects. This section, besides citing a number of disadvantages of unstructured testing and advantages of structured testing, also cites a few characteristics of the structured approach.

Disadvantages of unstructured testing

Unstructured testing is typified by a disorderly situation, in which it is impossible to predict the test effort, to execute tests feasibly or to measure results effectively. This is often referred to as 'ad hoc testing'. Such an approach employs no quality criteria in order to, for example, determine and prioritize risks and test activities. Neither is a test-design technique employed for the creation of test cases. Some of the findings that have resulted from the various studies of structured and unstructured testing are:

- Time pressures owing to:
  - absence of a good test plan and budgeting method
  - absence of an approach in which it is stated which test activities are to be carried out in which phase, and by whom
  - absence of solid agreements on terms and procedures for delivery and reworking of the applications.
- No insight in or ability to supply advice on the quality of the system due to:
  - absence of a risk strategy
  - absence of a test strategy
  - test design techniques not being used, therefore both quality and quantity of the test cases are inadequate.
- Inefficiency and ineffectiveness owing to:
  - lack of coordination between the various test parties, so that objects are potentially tested more than once, or even worse: not tested at all
  - lack of agreements in the area of configuration and change management for both test and system development products
  - the incorrect or non-use of the – often available – testing tools
  - lack of prioritization, so that less important parts are often tested before more risk-related parts.

Advantages of a structured testing approach

So what are the advantages, then, of structured testing? A simple, but correct, answer to that is that in a structured approach, the aforementioned disadvantages are absent. Or, put positively, a structured testing approach offers the following advantages:

- it can be used in any situation, regardless of who the client is or which system development approach is used
- it delivers insight into, and advice on, any risks in respect of the quality of the tested system
- it finds defects at an early stage
- it prevents defects

- the testing is on the critical path of the total development as briefly as possible, so that the total lead time of the development is shortened
- the test products (e.g. test cases) are reusable
- the test process is comprehensible and manageable.

Features of the structured testing approach

What does the structured testing approach look like? Many different forms are conceivable. In section 4.1.3 "The essentials of TMap NEXT®" and the subsequent chapters, the specific TMap form of this is given.
In general, it can be said that a structured testing approach is typified by:

- Providing a structure, so that it is clear exactly *what*, by *whom*, *when* and in *what sequence* has to be done.
- Covering the full scope and describing the complete range of relevant aspects.
- Providing concrete footholds, so that the wheel needn't be reinvented repeatedly.
- Managing test activities in the context of time, money and quality.

4.1.2 The role of testing

This section explains both the significance and role of certain test concepts in their environment. Spread across the following subjects, the associated concepts are explained:

- Testing and quality management
- Testing: how and by whom
- Test and system development process
- Test levels and responsibilities
- Test types

4.1.2.1 Testing and quality management

Quality was, is and remains a challenge within the IT industry. Testing is not the sole solution to this. After all, quality has to be built in, not tested in! Testing is the instrument that can provide insight into the quality of information systems, so that test results – provided that they are accurately interpreted – deliver a contribution to the improvement of the quality of information systems. Testing should be embedded in a system of measures in order to arrive at quality. In other words, testing should be embedded in the quality management of the organization.

The definition of quality as expressed by the ISO strongly hints at its elusiveness. What is clearly implied to one is anything but to another. Implicitness is very much subjective. An important aspect of quality management is therefore the minimization of implied requirements, by converting them into specified requirements and making the degree visible to which the specified requirements are met. The structural improvement of quality should take place top-down. To this end, measures should be taken to establish those requirements and to render the development process manageable.

Definition Quality assurance covers all the planned and systematic activities necessary to provide adequate confidence that a product or service meets the requirements for quality [ISO, 1994].

These measures should lead to a situation whereby:

- there are points of measurement and ratings that provide an indication of the quality of the processes (standardization)

- it is clear to the individual employee which requirements his work must meet and also that he can evaluate them on the basis of the above-mentioned standards
- it is possible for an independent party to evaluate the products/services on the basis of the above-mentioned standards
- the management can trace the causes of weaknesses in products or services, and consider how they can be prevented in future.

Preventive, detective and corrective measures are distinguished:
- Preventive measures are aimed at preventing a lack in quality. They can be, for example, documentation standards, methods, techniques, training, etc.
- Detective measures are aimed at discovering a lack of quality, for example by evaluation (including inspections, reviews, walkthroughs) and testing.
- Corrective measures are aimed at rectifying the lack of quality, such as the reworking of defects that have been exposed by means of testing.

It is of essential importance that the various measures are cohesive. Testing is not an independent activity; it is only a small cog in the quality management wheel. It is only one of the forms of quality control that can be employed. Quality control is in turn only one of the activities aimed at guaranteeing quality. And quality assurance is, in the end, only one dimension of quality management.

4.1.2.2 Testing, how and by whom

Testing often attracts little attention until the moment the test is about to begin. Then suddenly a large number of interested parties ask the test manager about the status. This section demonstrates, however, that testing is more than just the execution of tests. We then explain the ways of testing and by whom the testing can be carried out.

There is more to testing

Testing is more than a matter of taking measurements – crucially, it involves the right planning and preparation. Testing is the tip of the iceberg, the bigger part of which is hidden from view (see figure 18 "The iceberg").

Figure 18: The iceberg

In this analogy, the actual execution of the tests is the visible part, but on average, it only covers 40% of the test activities. The other activities – planning and preparation – take up on average 20% and 40% of the testing effort respectively. This part is not usually recognized as such by the organization, while in fact it is where the biggest benefit, not least regarding time, is to be gained. And, significantly, by carrying out these activities as

much as possible in advance of the actual test execution, the testing is on the critical path of the system development programme as briefly as possible. It is even possible, because of technical developments (test automation), to see a decreasing line in the percentage of test executions regarding preparation and planning.

Ways of testing

There are various ways of testing (in this case, executing tests). For example, is the testing being done by running the software, or precisely by not running it? And is a characteristic of the system being tested using test cases specially designed for it, or precisely not? A number of ways of testing are:

- Explicit testing
- Implicit testing

Explicit testing

With explicit testing, the test cases are explicitly designed to obtain information on the relevant quality characteristic. With the execution of the test, or the running of software, the actual result is compared against the expected result in order to determine whether the system is behaving according to requirements. This is the most usual way of testing.

Implicit testing

During testing, information can also be gleaned concerning other quality characteristics, for which no explicit test cases have been designed. This is called implicit testing. Judgments can be made, for example, on the user-friendliness or performance of a system based on experience gained without the specific test cases being present. This can be planned if there has been a prior agreement to provide findings on it, but it can also take place without being planned. For example, if breakdowns occur regularly during the testing. In that case, a judgment can be made concerning the security of company operations.

Who tests?

Anyone can do testing. Who actually does the testing is partly determined by the role or responsibility held by someone at a given time. This often concerns representatives from development, users and/or management departments. Besides these, testing is carried out by professional testers, who are trained in testing and who often bring a different perspective to testing. Where, for example, a developer wants to demonstrate that the software works well ("Surely I'm capable of programming?"), the test professional will go in search of defects in the software. Moreover, a test professional is involved full-time in testing, while the aforementioned department representatives in many cases carry out the testing as a side issue. In practice, the mix of well-trained test professionals and representatives from the various departments leads to fruitful interaction, with one being strong in testing knowledge and the other contributing much subject or system knowledge.

4.1.2.3 Test and system development process

The test and system development processes are closely intertwined. One delivers the products, which are evaluated and tested by the other. A common way of visualizing the relationship between these processes is the so-called V model. A widely held misunderstanding is that the V model is suited only for a waterfall method. But that misrepresents the intention behind the model. It is also eminently usable with an iterative and incremental system development method. Therefore, with such a method, a V model can be drawn, for example, for each increment. Many situations are conceivable that

influence the shape and the specific parts of the V model. A few situations are shown in the box below: "Influences on the V model".
With the help of the V model, the correlation between test basis, evaluation and testing (test levels) is explained in this and the following subsection.

In more detail

Influences on the V model
The form and specific parts of a V model can vary through, for example:

- The place of the testing within the system development approach.
  - Using a waterfall development method with characteristics including: construction of the system in one go, phased with clear transfer points, often a lengthy cyclical process (SDM, among others).
  - Using an incremental and iterative development method with the following possible characteristics: constructing the system in parts, phased with clear transfer points; short cyclical process (DSDM and RUP, among others).
  - Using an agile development method characterized by the four principles: individuals and interaction over processes and tools, working software over extensive system documentation, user's input over contract negotiation, reacting to changes over following a plan (extreme programming and SCRUM, among others).
- The place of testing within the life cycle of the information system.
  - Are we looking at new development or the maintenance of a system?
  - Does this involve the conversion or migration of a system?
- A self-developed system, a purchased package, purchased components, or distributed systems.
- The situation whereby (parts of) the system development and/or (parts of) the testing are outsourced (outsourcing and off-/near shoring, among other things).

Left side of the V model

In figure 19 "V model (the left side)" the left-hand sideshows the phases in which the system is built or converted from wish, legislation, policy, opportunity and/or problem into the solution that has been realized. In this case, the left-hand side shows the concepts of requirements, functional and technical designs and realization. While the exact naming of these concepts is dependent on the selected development method, it is not required in order to indicate the relationship between the system development and test process at a general level.

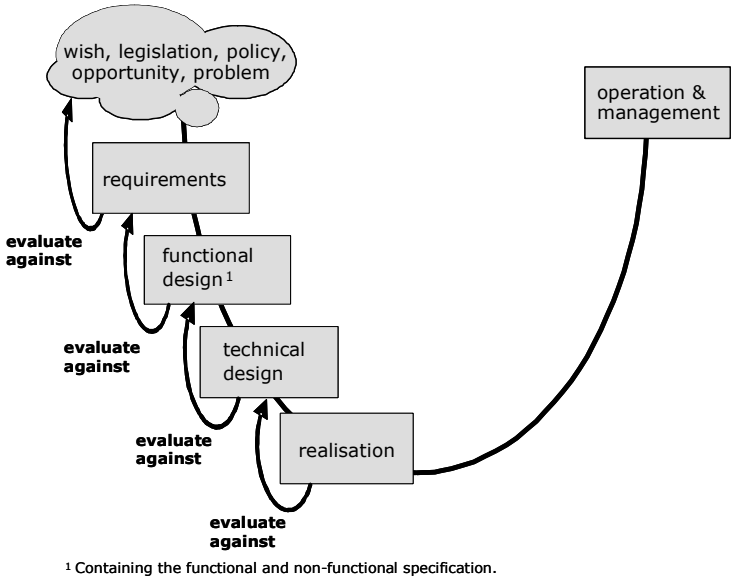


Figure 19: V model (the left side)

Evaluation

During the system development process, various interim and end products are developed. Depending on the selected method, these take a particular form, content and relationship with each other and can be tested on these.

Definition Evaluation is assessing the products in the system development process without running software.

In the V model, the left-hand side shows which interim products can be evaluated (against each other). In evaluation, the result can be compared with:

- The preceding interim product
  For example, is the functional design consistent with the technical design?
- The requirements from the succeeding phase
  For example, can the builder realize the given design unambiguously and are the specifications testable?
- Other interim products at the same level
  For example, is the functional design consistent internally and with functional designs related to it?
- The agreed product standard
  For example, are there use cases present?
- The expectations of the client (see box "Realized requirements")
  Is the interim product still consistent with the expectations of the acceptors?

Besides interim products, end products can be evaluated as well. For instance by inspecting documentation like safety procedures, courses, manuals etc.

With this, various techniques are available for the evaluation: reviews, inspections and walkthroughs (see also section 4.12 "Evaluation techniques").

In more detail

Realized requirements
What about the trajectory of wish, legislation, etc., to product? Will, for example, all the requirements be realized, or will something be lost along the way? A survey carried out by the Standish Group unfortunately shows a less than encouraging picture. The findings of the survey (figure 20 "Realized requirements"), in which the percentage of realized requirements was determined, shows that, of the original defined requirements, only 42% to 67% are actually realized by the project [The Standish Group, 2003].

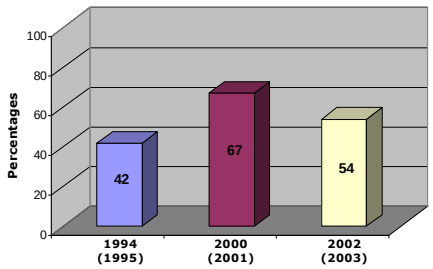


Figure 20: Realized requirements

Besides normal evaluation results (the finding of defects) a well-organized and executed evaluation process can deliver a contribution to a higher realization percentage in respect of the original defined requirements.

4.1.3 The essentials of TMap NEXT®

This chapter describes the specific TMap content of a structured test method. The content can be summarized in four essentials.
The four essentials of TMap:
1. TMap is based on a business-driven test management (BDTM) approach.
2. TMap describes a structured test process.
3. TMap contains a complete tool box.
4. TMap is an adaptive test method.

The first essential can be related directly to the fact that the business case of IT is becoming ever more important to organizations. The BDTM approach provides content that addresses this fact in TMap and can therefore be seen as the 'leading thread' of the structured TMap test process (essential 2). The TMap life cycle model is used in the description of the test process. Furthermore various aspects in the field of infrastructure, techniques and organization must be set up to execute the test process correctly. TMap provides a lot of practical applicable information on this, in the form of e.g. examples, checklists, technique descriptions, procedures, test organization structures, test environments and test tools (essential 3). TMap also has a flexible setup so that it can be implemented in different system development situations: both for new development and maintenance of a system, for a self-developed system or an acquired package, and for outsourcing (parts of) the testing process. In other words, TMap is an adaptive method (essential 4).

In the "TMap model of essentials" below, the left triangle symbolizes BDTM, the triangle at the bottom the tool box, the parallelogram the structured test process, and the 'circle' TMap's adaptiveness.

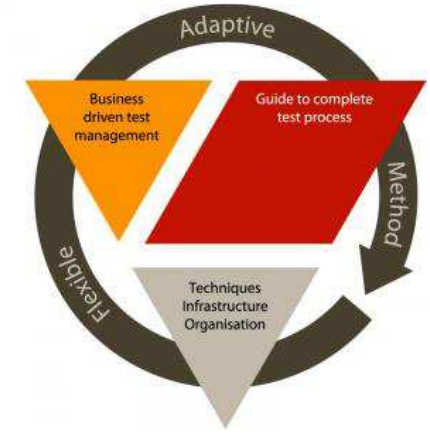


Figure 21. TMap model of essentials.

4.1.3.1 Business Driven explained

The key to testing is that tests are executed on the basis of test cases, checklists and the like. But what kind of tests are they? To ensure the tests' usefulness, they must be set up to test those characteristics and parts of a test object that represents a risk if it does not function adequately in production later on. This means that various considerations have already been made before test execution can begin. In other words, some thought has already been given to which parts of the test object need not be tested, and which must be tested and how and with what coverage. So what determines this? Why not test all parts of the test object as thoroughly as possible? If an organization possessed unlimited resources, one option might indeed be to test everything as thoroughly as possible. But naturally, in real life an organization rarely has the resources to actually do this, which means that choices must be made in what is tested and how thoroughly. Such choices depend on the risks that an organization thinks it will incur, the available quantities of time and money, and the result the organization wishes to achieve. The fact that the choices are based on risks, result, time and cost is called business-driven and constitutes the basis for the BDTM approach. To understand and apply the BDTM approach, we first explain the concept of the "business case".

Business case as determining factor

IT projects must be approach increasingly from a purely economic perspective. The theory of IT governance controls projects on the basis of four aspects: result, risk, time and cost. For instance, it might be a more attractive investment for an organization to start a high-risk project that potentially yields a high result than a project with very low risks where the benefits barely exceed the costs.

Normally, a business case is at the basis of an IT project. There are various definitions of business case, including the project-oriented one below according to [PRINCE2, 2002].

Definition
The business case provides the economic justification for the project and answers the questions: *why* do we do this project, *which* investments are needed, *what* does the client wish to achieve with the result?

During the project, the business case is verified at predefined points in time to ensure that the eventual results remain valid for the client. TMap supports the economic justification of

IT, translating it to the activity of testing. TMap assumes that a project approach based on a business case complies with the following characteristics:

- The approach focuses on achieving a predefined result.
- The total project to achieve this result is realized within the available (lead) time.
- The project to achieve this result is realized at a cost in balance with the benefits the organization hopes to achieve.
- The risks during commissioning are known and as small as possible. All of this within the framework set by the abovementioned characteristics.

The four IT governance aspects described above can be found in these characteristics. For the successful execution of a project, it is important that the test process is aligned with the business case. The relationship between the business case and the test process is made via the business-driven test management approach. In other words, with this approach, the business case characteristics can be 'translated' to the test process.

Characteristics of a business-driven test management approach

Often test plans and reports fail to appeal to the client. The reason being that in the past the tester virtually always made decisions from an IT perspective. The test process was internally oriented and filled with test and IT jargon. This made it difficult to communicate with a non-IT client, such as a user department, even though this is extremely important.

TMap devotes explicit attention to communication due to the business-driven test management approach[4]. BDTM starts from the principle that the selected test method of operation must enable the client to control the test process and (help) determine the test method of operation. This gives the testing an economic character. The required information to make this possible is delivered from the test process.

BDTM has the following specific properties:

- The total test effort is related to the risks of the system to be tested for the organization. The deployment of people, resources and budget thereby focuses on those parts of the system that are most important to the organization. In TMap, the test strategy in combination with the estimated effort is the instrument to divide the test effort over system parts. This provides insight into the extent to which risks are covered, or not.
- The estimated effort and planning for the test process are related to the defined test strategy. If changes are implemented that have an impact on the thoroughness of testing for the various system parts or systems, this is translated immediately to a change in the estimate and/or planning. The organization thus is ensured of an adequate view of the required budget, lead time and relationship with the test strategy at all times.
- At various moments in the testing programme, the client is involved in making choices. The advantage is that the test process matches the wishes and requirements – and therefore the expectations – of the organization as adequately as possible. Moreover, BDTM provides handholds to visualize the consequences of future and past choices explicitly.

The steps in the business-driven test management approach

To understand the BDTM approach, it is important to keep an eye on the final objective. Which is to provide a quality assessment and risk recommendation about the system. Since not everything can ever be tested, a correct assessment can only be realized by dividing

---

[4] Please note that BDTM is not an entirely accurate name. The word "business" suggests that it is intended exclusively for the link with the user departments, while testers clearly often still deal exclusively with IT departments. In this book, however, the general name BDTM is used.

the test effort, in terms of time and money, as adequately as possible over parts and characteristics of the system to be tested. The steps of BDTM focus on this (see figure 22):

1. Formulating the assignment and gathering test goals
   In consultation with the client, the test manager formulates the assignment, taking account of the four BDTM aspects: result, risk, time and cost.

   The test manager gathers the test objectives to determine the desired results of testing for the client. A test goal is a goal for testing relevant to the client and other acceptors, often formulated in terms of IT-supported business processes, realized user requirements or use cases, critical success factors, change proposals or defined risks (i.e., the risks to be covered).

2. Determining the risk class for each combination of characteristic and object part.
   When *multiple* test levels are involved, it is determined in a plan which test levels must be set up (master test plan). It is often already determined on the basis of a product risk analysis[5] *what* must be tested (object parts) and *what* must be investigated (characteristics).

   If only *one* test level is involved, or if no or an overall product risk analysis was performed at the master test plan level, a (possibly supplementary) product risk analysis is performed within the relevant test level.

   The eventual result (whether it is arrived at immediately or after one or more supplementary analyses) is a risk table defining a risk class related to the test goals and the relevant characteristic per object part ("Master test plan risk table").

   A table then provides a guideline for the relative test intensity per combination of characteristic/object part and test level ("Master test plan strategy table").

Now an *iterative process* emerges:

3. Determining whether a combination of characteristic and object part must be tested thoroughly or lightly.
   To determine the thoroughness of testing, the risk class per object part determined in the previous step is used as a starting point. Initially, the following applies: the greater the risk, the more thorough the required testing. The result is recorded in a strategy table per test level ("Test plan strategy table").

4. An overall estimate is then made for the test and a planning set up. This is communicated with the client and other stakeholders and, depending on their views, adjusted as necessary. In this case, steps 3 and 4 are executed once again. This emphatic gives the client control of the test process, enabling him to manage based on the balance between result and risk on the one hand and time and cost on the other.

End of iteration.

5. Allocating test techniques to the combinations of characteristic and object part.

---

[5] A product risk analysis (PRA) aims to ensure that the various stakeholders and test manager achieve a joint view of the more and less high-risk parts/characteristics of the system. The focus in the PRA is on the product risks, i.e. what is the risk to the organization if the product does not have the expected quality?

When the client and stakeholders agree on the estimate and the planning, the test manager completes a "Test design table". In here, the decisions concerning thorough and less thorough testing are translated to concrete statements about the targeted coverage. He then allocates test techniques to the combinations of characteristic and object part. The available test basis, among other things, is taken into account. These techniques are used to design and execute the test cases (and/or checklists) at a later stage. This is where the primary test process starts.

6. Throughout the test process, the test manager provides the client and other stakeholders with adequate insight into and control options over:
   - the progress of the test process
   - the quality and risks of the test object
   - the quality of the test process.

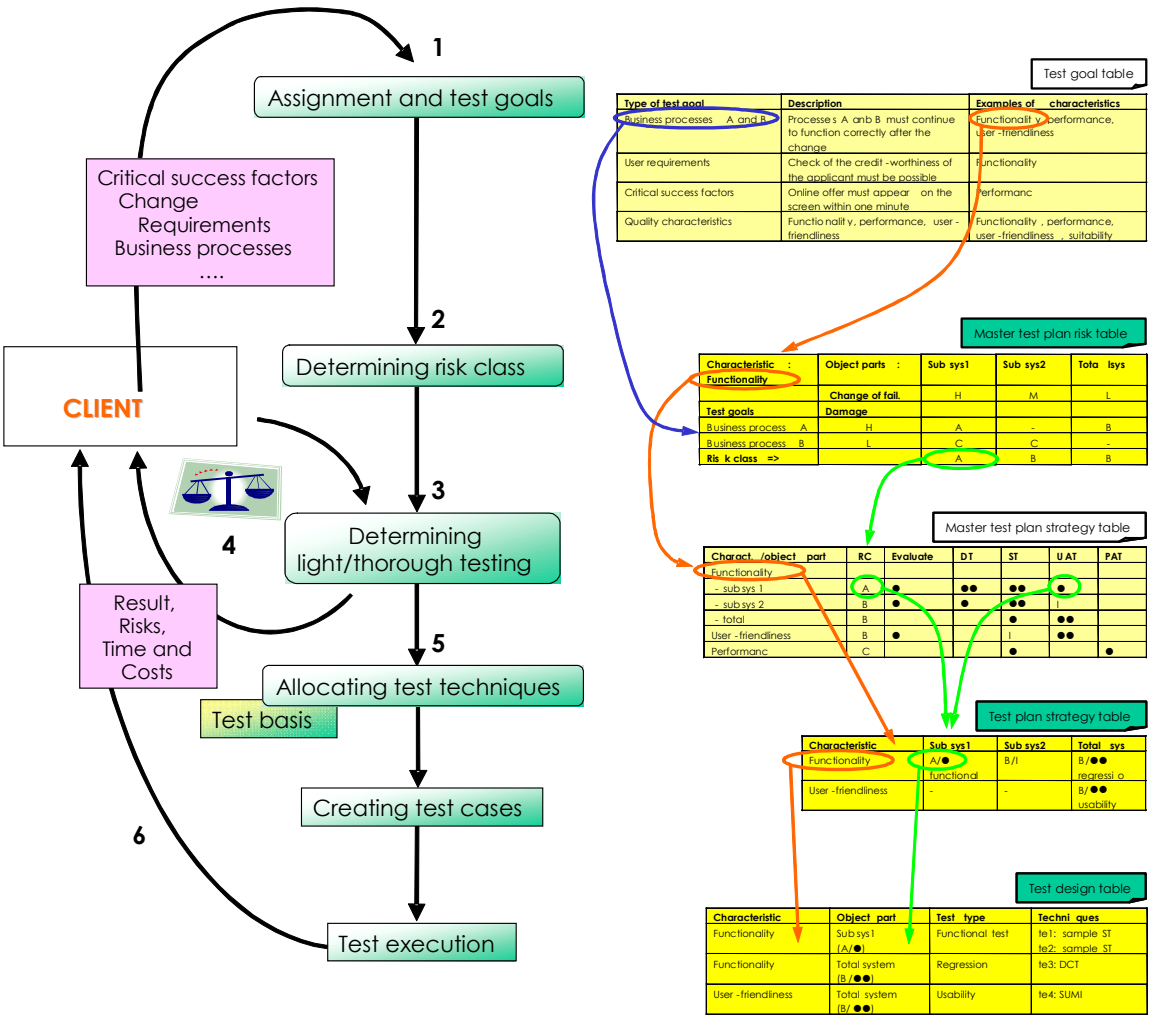


Figure 22: BDTM steps

In summary, the advantages of the BDTM approach are:

- The client having control over the process.
- The test manager communicates and reports in the terminology of the client with information that is useful in the client's context. E.g. by reporting in terms of test goals (such as business processes) instead of object parts and characteristics.

- At the master test plan level, detailing can be as intensive as required or possible. This may enable expending less effort on performing a product risk analysis or creating a test strategy for the separate test levels, or even to skip these steps (explanation of master test plan in subsequent section).

4.1.3.2 Structured test process

This section describes the phasing and activities in the following TMap processes:
- Master test plan, managing the total test process
- Acceptance and system tests
- Development tests.

Master test plan and other TMap NEXT processes

When the test manager, after consultation with the receiving parties, decides what will be tested for each test level, chances are that in the total picture of testing, certain matters will be tested twice unnecessarily. Or that certain aspects are ignored. The method should therefore be vice versa. A test manager, in consultation with the client and other stakeholders, makes a total overview of the distribution across test levels as to what must be tested when and with what thoroughness. The aim is to detect the most important defects as early and economically as possible. This agreement is defined in the so-called master test plan (MTP). This plan constitutes the basis for the test plans for the separate test levels. In addition to this content-based alignment, other types of alignment are: ensuring uniformity in processes (e.g. the defect procedure and testware management), availability and management of the test environment and tools, and optimal division of resources (both people and means) across the test levels.

This means that in addition to test levels like acceptance, system and development tests, the master test plan also plays an important part in TMap. Both for the master test plan and the test levels, it is important to set up a good process for creating plans and preparing, executing and managing activities.

While the goals of the acceptance and system tests differ, these test levels are not described separately, but as one single process. This was decided because the activities in both test levels are virtually the same and separate process descriptions would therefore have (too) much overlap.

In addition to these processes, the process "Supporting processes" has been defined because it is more efficient to organize certain aspects/support centrally than per project. This involves supporting processes for the following subjects:
- Test policy
- Permanent test organization
- Test environments
- Test tools
- Test professional.

The supporting processes are discussed in relevant places as part of the complete tool box (see the subsection 'Complete tool box').

Process: master test plan, managing the total test process

The master test plan provides insight into the various test and evaluation levels to be used, in such a way that the total test process is optimized. It is a management tool for the underlying test levels.

The process "Master test plan, managing the total test process" is split up into two phases: the Planning phase of the total test process and the Control phase of the total test process.

Planning phase of the total test process
The author of the MTP, often the test manager formulates the assignment, taking into account the four BDTM aspects of result, risks, time and cost, in consultation with the client. The test manager then works on the upcoming programme by having discussions with stakeholders and consulting information sources, such as documentation. In parallel, the test manager further elaborates the assignment and determines its scope in consultation with the client. In this phase, the first four steps of BDTM are executed: performing a PRA, establishing a test strategy, estimate and planning (see figure 22 "BDTM steps").

Further activities in the creation of the plan are: the test manager defines the products that must be delivered by the test levels and makes a proposal as to the setup of the test organization, centrally and overall per test level. The test manager aligns the infrastructure requirements of the test levels in order to deploy the – often scarce – test infrastructure as efficiently as possible. Test management can also be set up in part at the master test plan level. This can be achieved both by defining central procedures and standards for management and by the central management of certain aspects. Both options aim to prevent reinventing the wheel in the various test levels. The main risks threatening the test process are listed, and possible measures are proposed to manage these risks. As his last step, the test manager submits the master test plan to the client for approval.

Control phase of the total test process
The aim of this activity is controlling the test process, infrastructure and test products at the overall level to provide continuous insight into the progress and quality of the total test process and the quality of the test object. Conformable to the frequency and form defined in the test plan, reports are made on the quality of the test object and the progress and quality of the test process. From the very first test activities, the testers develop a view of that quality. It is important that this is reported in every stage of the test process. The client receives periodical reports, and ad-hoc reports on request, on the condition of the system. Such reporting and adjustment are a vital part of the BDTM approach (BDTM step 6) and take place at both the level of the master test plan and that of the test level (see figure 23).

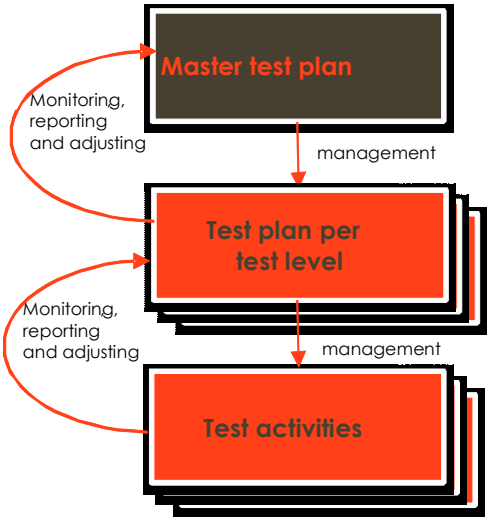


Figure 23. Execution, monitoring, reporting and adjusting.

Process: acceptance and system tests

See section 4.1.5.

Process: development tests

See section 4.1.6.

4.1.3.3 Complete tool box

TMap supports the correct execution of the structured test process with a complete tool box. The tool box focuses on working with the following subjects:

- Techniques : *how* it is tested
- Infrastructure : *where* and *with what* it is tested
- Organization : *who* does the testing

The various tools are described in more detail in the TMap Suite at the moment they can be used. With the tool box, the tester possesses a great number of options to meet the test challenge successfully.

Techniques

Many techniques can be used in the test process. A test technique is a combination of actions to produce a test product in a universal manner.

TMap provides techniques for the following:

- Test estimation
- Defect management
- Creating metrics
- Product risk analysis
- Test design
- Product evaluation.

TMap also offers various checklists and overviews that can be used as a tool during the preparation and/or execution of certain activities.

The (groups of) test techniques are summarized below.

Test estimation

Estimates can be made at a number of different levels. The various estimation levels are shown in figure 24.

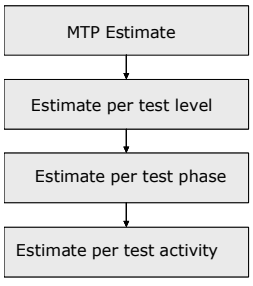


Figure 24. Estimation levels.

Independent of the level, creating an estimate consists of the following generic steps:

1. Inventory the available material that can serve as a basis for the estimate.
2. Select (a number of) estimating techniques.
3. Determine the definitive estimate.
4. Present the outcome.

Choosing the estimating techniques in particular is a step requiring experience. You can select from several estimating techniques:

- Estimation based on ratios. Here, the test effort is generally measured against the development effort, e.g. in percentage ratios.
- Estimation based on test object size.
- Estimation using a 'Work Breakdown Structure'.
- Proportionate estimation based on the total test budget.
- Estimation on the basis of extrapolating experience figures from the beginning of the testing programme.
- Estimation on the basis of size and strategy using TMap's test point analysis (TPA).

Furthermore, TMap provides a technique to create an evaluation estimate.

Defect management

A defect is an observed difference between the expectation or prediction and the actual outcome. While the administration and monitoring of the defects is factually a project matter and not one of the testers, testers are usually very closely involved. A good administration must be able to monitor the lifecycle of a defect and provide various overviews. These overviews are used, among other things, to make well-founded quality statements. See section 4.7.

Creating metrics

The definition, maintenance and use of metrics is important to the test process because it enables the test manager an answer, supported by facts, to questions like:

- What about the quality of the test object?
- What about the progress of the test process?

A structured approach to realize a set of test metrics is using the Goal-Question-Metric (GQM) method.
In addition to describing the GQM method, TMap gives instructions to set up a practical test metrics starter set. It also provides a checklist that can be useful to make pronouncements on the quality of the object to be tested and the quality of the test process.

Product risk analysis

A product risk analysis (PRA) is analyzing the product to be tested with the aim of achieving a shared view, among the test manager and other stakeholders, of the more or less risky characteristics and components of the product to be tested so that the thoroughness of testing can be agreed upon. The focus in PRA is on the product risks, i.e. what is the risk to the organization if the product does not have the expected quality?

The result of the PRA constitutes the basis for the subsequent decisions in strategy as to light, thorough or non testing of a characteristic (e.g. a quality characteristic) or object part (component) of the product to be tested.

Test design

See chapter 3.

Product evaluation

See section 2.20 (Building Block 20: Reviewing requirements).

Various checklists and overviews

TMap offers a great variety of checklists that will constitute a welcome addition to the tester when executing certain activities. For instance, there are checklists that can be used as support in taking stock of the assignment, determining the test facilities, determining the test project risks, establishing the test strategy, the evaluation of the test process, taking interviews, and determining whether adequate information is available to use a specific test design technique. TMap also offers other tools, such as an overview matrix of automated tools per TMap activity, a test type overview, and criteria to select a tool.

These tools and many more can be found on and downloaded from www.tmap.net.

Infrastructure

Test environments, test tools and workplaces are necessary to execute tests..

Test environments

See also section 4.5.

A fitting test environment is necessary for testing a test object (running software). A test environment is a system of components, such as hardware and software, interfaces, environmental data, management tools and processes, in which a test is executed. The degree to which it can be established in how far the test object complies with the requirements determines whether a test environment is successful. The setup and composition of a test environment therefore depend on the objective of the test. However, a series of generic requirements with which a test environment must comply to guarantee reliable test execution can be formulated. In addition to being representative, manageable and flexible, it must also guarantee the continuity of test execution.

Setting up and managing the test environment represents an expertise of which testers generally have no knowledge. This is why a separate department – outside the project – is generally responsible for setting up and managing the test environment.

Test tools

See also section 2.16 and section 4.6.

To execute the tests efficiently, tools in the form of test tools are necessary. A test tool is an automated instrument that provides support to one or more test activities, such as planning and control, test specification, and test execution. The use of tools can have the following advantages:

- Increased productivity
- Higher testing quality
- Increased work enjoyment
- Extension of test options.

The test tools are classified in four groups:

- Tools for planning and managing the test
- Tools for designing the test
- Tools for executing the test
- Tools for debugging and analyzing the code.

Workplaces

One of the aspects that is often forgotten in testing, is the availability of a workplace where testers can do their job under good conditions, effectively and efficiently. This means office setup in the broadest sense since the testers must also be able to do their work under good conditions. The workplace is therefore more than just office space and a PC. Matters such as access passes, power supply and facilities to have lunch must be arranged. At first sight, the workplace for a tester does not differ much from the regular workplace. But appearances can be deceptive. What is tested is often new to the organization and the workplace. Testers may have to deal with the situation that their workplace is not yet prepared for the new software. For example, testers often require separate authorizations. They must, for instance, be able to install the new software on their local PC. This may also be necessary for the use of certain test tools.

Organization

See section 2.4 (Building Block 4: Test Organization), section 2.13 (Building Block 13: Permanent Test Organization) and section 2.18 (Building Block 18: Integrated Test Organization).

4.1.3.4 Adaptive and complete method

TMap is an approach that can be applied in all situations and in combination with any system development method. It offers the tester a range of elements for his test, such as test approaches, coverage types, test design techniques, test infrastructure, test strategy, phasing, test organization, test tools, etc. Depending on the situation, the tester selects the TMap elements (Building Blocks) that he will deploy. There are situations in which only a limited number of elements need to be used; but in other situations he will have to use a broad range of elements. This makes TMap an adaptive method, which in this context is defined as:

Definition

> *Adaptive is the ability to split up an element into sub-elements that, in a different combination, result in a new, valuable element for the specific situation.*

The adaptiveness of TMap is not focused on a specific aspect of the method, but is embedded throughout the method. Adaptiveness is more than just being able to respond to the changing environment. It is also being able to leverage the change to the benefit of testing. This means that TMap can be used in every situation and that TMap can be used in a changing situation. In the course of projects and testing, changes may occur that have an impact on earlier agreements. TMap offers the elements to deal with such changes.

TMap's adaptiveness can be summarized in four adaptiveness properties:

- *Respond* to changes
- *(Re)use* products and processes
- *Learn* from experiences
- *Try* before use

These properties are explained in further detail below.

Respond to changes

Adaptiveness starts with determining the changes and responding to them. In TMap, this happens from the very beginning in the earliest activities of the (master) test plan. When determining and taking stock of the assignment, obtaining insight into the environment in which the test is executed and establishing possible changes play a major part. This is precisely where the basis is created for the further elaboration and implementation of the method. Which test levels, test types, phases, and tools are used and how? But it is not limited to these activities. The test strategy and associated planning are defined in close consultation with the client. If the test strategy and derived estimate and planning are not acceptable to the client, the plan is adapted. This emphatic gives the client control of the test process, enabling him to manage based on the balance between result and risk on the one hand and time and cost on the other. Such feedback is provided throughout the testing programme, and in the control phase, the test manager may also decide to adapt certain aspects of the test plan in consultation with the client.

(Re)use products and processes

Being able to use products and processes quickly is a requirement for adaptiveness. TMap offers this possibility, among other things thanks to the large quantity of tools included in the form of test design techniques, checklists, templates, etc. These can be found in the book and on www.tmap.net.

In addition to use, reuse plays an important part. The emphasis in this respect lies in the Completion phase, where the activities are defined to identify what can be reused and how it can be optimally preserved. TMap offers various forms of a permanent test organization for the organizational anchoring of the reuse of products and processes.

Learn from experiences

As a method, TMap offers the space to learn and apply what was used. Therefore the activity evaluating the test process is incorporated into the test process. Another important instrument is the use of metrics. For the test process, metrics on the quality of the test object and the progress and quality of the test process are extremely important. They are used to manage the test process, justify the test recommendations, and compare systems or test processes. Metrics are also important to improve the test process through assessing the consequences of certain improvement measures.

Try before use

TMap offers room to try something before it is actually used. The main instruments here are the activities relating to the intake. The intake of the test basis (using a testability review), of the test infrastructure, and of the test object allow one to try first before actually using.

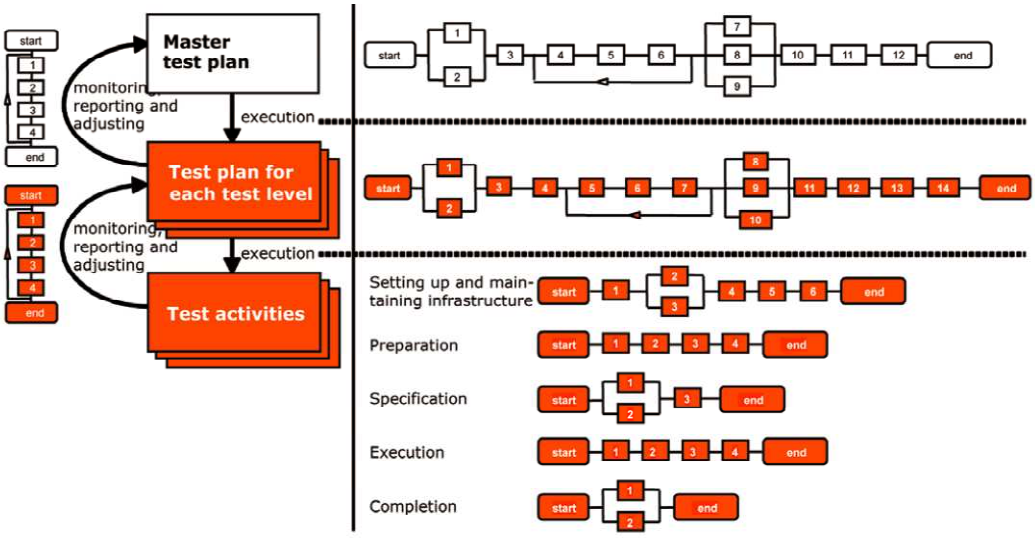
Implementing TMap does not mean that everything in the TMap Suite should be used without question. Another form of trying before using is therefore 'customizing' TMap to fit a specific situation. A selection can be made from all of the TMap Building Blocks to achieve this. After the approach, customized to the situation, has been tried out ('pilot'), it can be rolled out in the organization.

For many situations, 'customizing' TMap has already been done. The specific TMap recipe for a certain situation is called a "pattern". TMap explicitly invokes the development of new patterns, based on existing and/or new Building Blocks.

4.1.4 Testing in an agile environment

The TMap phases turn out to be easily integrated with the scrum model. But how to integrate the activities of those phases? Practice shows that these, too, can be adapted for use in a scrum approach without too much trouble.

The following sections explain how, per phase, the most important activities in a scrum approach can be executed. Please keep in mind that scrum is an agile approach, whereas TMap is an adaptive approach. This means that all suggestions and practical examples mentioned in the following sections will probably have to be adapted to your own specific situation. In case you do not have the TMap NEXT book at hand, figure 25 will provide you with a complete overview of the TMap phases and activities.



Planning the total test process
1. Establishing the assignment
2. Understanding the assignment
3. Analyzing the product risks
4. Determining the test strategy
5. Estimating the effort
6. Determining the planning
7. Defining the test products
8. Defining the organization
9. Defining the infrastructure
10. Organizing the management
11. Determining test process risks and countermeasures
12. Feedback and consolidation of the plan

Control of the total test process
1. Management
2. Monitoring
3. Reporting
4. Adjusting

Planning each test level
1. Establishing the assignment
2. Understanding the assignment
3. Determining the test basis
4. Analyzing the product risks
5. Determining the test strategy
6. Estimating the effort
7. Determining the planning
8. Allocating test units and test techniques
9. Defining the test products
10. Defining the organization
11. Defining the infrastructure
12. Organizing the management
13. Determining test process risks and counter measures
14. Feedback and consolidation of the plan

Control of each test level
1. Management
2. Monitoring
3. Reporting
4. Adjusting

Setting up and maintaining infrastructure
1. Specifying the infrastructure
2. Realizing the infrastructure
3. Specifying the infrastructure intake
4. Intake of the infrastructure
5. Maintaining the infrastructure
6. Preserving the infrastructure

Preparation
1. Collection of the test basis
2. Creating checklists
3. Assessing the test basis
4. Creating the testability review report

Specification
1. Creating test specifications
2. Defining central starting point(s)
3. Specifying the test object intake

Execution
1. Intake of the test object
2. Preparing the starting points
3. Executing the (re-)tests
4. Checking and assessing the test results

Completion
1. Evaluating the test process
2. Preserving the testware

Figure 25. TMap NEXT phases and activities.

Planning

Just as in the planning phase of TMap, the planning schedule in scrum is also executed at various moments: at the start of the project, at the beginning of each sprint and during the daily scrum (see figure 26). The formulation of a test strategy is an important — but not the

only — activity of the planning activities, and that is why the moments at which test strategy formulation takes place are separately mentioned in this section: the formulation of the project test strategy at the start of the project and the formulation of the sprint test strategy at the beginning of the sprint. The daily scrum discusses the planning of the tasks that will be worked at that day and priorities are adjusted if necessary.

Figure 26. Planning activities.

Project test strategy

The planning schedule for the total test process is formulated at the beginning of a project. However, scrum is not primarily concerned with the actual planning of tasks in terms of fixed points in time. These are placed on a scrum board and become 'active' when the time is right. At this point in time of the scrum project, the focus lies more on defining a global test strategy. This is sometimes referred to as a 'project test strategy' — in a traditional developmental environment this would be a component of the master test plan.

It is good to be aware that the formulation of the project test strategy takes place during the 'planning' scrum event, and is incorporated into the project planning (product backlog) schedule. And in scrum projects that begin with a sprint 0, this is the moment to determine the high-level product risk and the project test strategy. This occurs parallel to other sprint 0 'setting up activities' such as arranging a kick-off, setting up tools, determining a definition of done, setting up the process, estimating the effort, formulating a communications plan and providing the training (see figure 27).

These activities are only examples of activities that take place in a sprint 0, and can and will differ per organization. In content-related terms, the project test strategy must be compact and global. After all, the strategy will alter during the various sprints in the project due to the ongoing acquisition of insight.

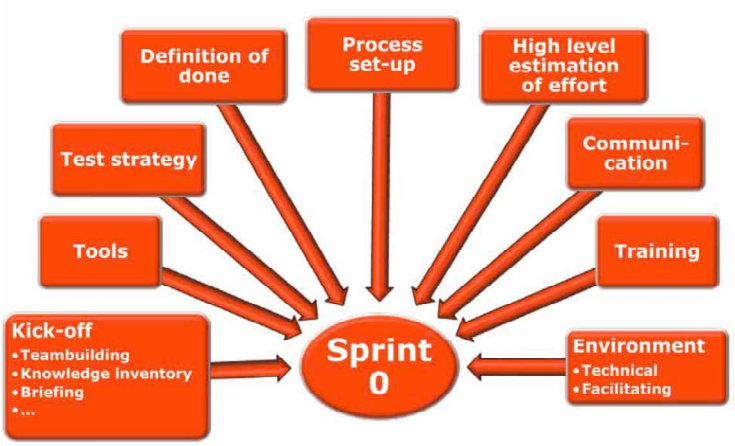


Figure 27. Possible activities in Sprint 0.

The most important themes to include in the project test strategy are the product risks for each backlog item and a global test strategy covering all the sprints (see Figures 15 and 16). After all, a scrum project always starts with a planning schedule.
To do this properly, it is essential to gain insight into the product risks and the required test intensity, or, in other words, the degree to which the risks have to be covered. High risks and thorough testing will ultimately have different outcomes for the scrum planning schedule than low risks and light testing will, and this may have consequences for the prioritization of the product backlog items.

Themes such as planning and estimating are components of the above-mentioned scrum planning schedule and demand little or no attention in the project test strategy. The assignment to perform the tests is given by the product owner and can be translated into, for example, criteria that are included in the definition of done. This aspect, too, requires very little attention in the project test strategy.

Defining the test infrastructure (including other test environments, test data and test tools, etc.) could be incorporated into another plan, but if a sprint 0 is scheduled it is better to configure the test infrastructure immediately and — inasmuch as it may be necessary — to have it ready and available, or at least to make a start on it before the first sprint begins. If no sprint 0 has been scheduled, these tasks can also be included on the backlog as technical product backlog items.

Sprint test strategy

The sprint test strategy (product risks and test strategy) is defined during the sprint planning event and included in the sprint backlog. For example, during the sprint planning stage, the development team estimates, with the aid of planning poker, the amount of time required for each task of a sprint backlog. The amount of time required is influenced by factors such as whether the test has to be thorough or light, which, in turn, is related to the product risk. Therefore, it is advisable to include the risk classification of a backlog item — particularly in the case of user stories — in addition to the priority specified by the product owner, before the planning poker is initiated.

The sprint test strategy is a detailed infill of the project test strategy and is liable to change in the course of the sprint. Similar to the project test strategy, this must also be a compact strategy, preferably one that fits on a whiteboard. The most important components are the

product risk analysis and the test strategy for the current sprint. In addition to the overall product risk analysis, the team is the most important source for the product risk analysis.

The risk analysis described in TMap can be adapted and executed as follows. Of course, all members of the scrum team are present here and actively participate. In view of the fact that the analysis takes place per sprint, no huge quantities of backlog items are involved, so that the analysis need not be very time-consuming. An hour often turns out to be more than sufficient. The execution can be done by means of the following steps:

1. Gather the scrum team members together.
2. List all the backlog items of the current sprint on a whiteboard.
3. Ask all team members individually which quality characteristics of each backlog item are important to them and ask if anything should be added to the list.
4. Determine the possible damage and chance of failure for each combination of backlog item and quality characteristic. The product risk is then: damage × chance of failure.

The risk table could look something like figure 28.

Item	Characteristic	Damage	Chance of failure	Risk class
A	Functionality	3	3	9
	Usability	2	1	2
B	Functionality	2	2	4
	Security	3	2	6
C	Functionality	2	1	2
D	Performance	2	1	2
E	Performance	1	1	1
F	Functionality	2	2	4
	Suitability	2	2	4
..	..	..	..	..

Figure 28. Risk table.

The test strategy must be determined in the next step, which specifies the test intensity with which a combination of backlog item and quality characteristic is to be tested. To make the test practically applicable, columns such as 'test intensity' and 'test design technique' can simply be added. This helps the team with a test role in the specification of the test cases (see figure 29). Then 'simply' apply the test design technique to the backlog item, on each line in the table, and the test intensity — depth of testing — in relation to the risk to be covered is achieved.

5. Determine the test intensity.
6. Determine on the basis of test intensity and quality characteristic which test design technique should be applied.

Item	Characteristic	Damage	Chance of failure	Risk class	Intensity	Test design technique
A	Functionality	3	3	9	•••	MCC
	Usability	2	1	2	•	SYN
B	Functionality	2	2	4	••	ET
	Security	3	2	6	••	SEM-MCDC
C	Functionality	2	1	2	•	DCoT-EQ
D	Performance	2	1	2	•	EG
E	Performance	1	1	1	•	EG
F	Functionality	2	2	4	••	ECT-MCDC
	Suitability	2	2	4	••	PCT-TDL2
..	..	..	..	..		

Figure 29. Test strategy table.

ET : Exploratory Testing
ECT-MCC : Elementary Comparison Test- Multiple Condition Coverage,
SYN : Syntactic Test,
ECT-MCDC : Elementary Comparison Test - Modified Condition/Decision Coverage,
SEM-MCDC : Semantic Test - Modified Condition/Decision Coverage,
DCoT-EC : Data Combination Test – Equivalence Classes,
EG : Error Guessing,
PCT-TDL2 : Process Cycle Test – Test Depth Level 2
Want to know more? Consult chapter 3 ("Website").

The test design techniques are listed in the test strategy table. Of course, other quality measures are also possible, such as evaluation (carrying out a review for example or an inspection), pair programming or the performance of a more severe unit test. You can adapt the table completely to your own situation. If, for example, you wish to widen the concept dealt with in the column 'Test design technique', you can simply change the column to 'Quality measures' for example. A column entitled 'Moment/Location', for instance, can also be added if considered relevant.

Now that the test strategy table has been completed, it is now time to record the rows in the table as tasks on the scrum board. Accordingly, the test strategy table does not remain a purely 'stand-alone' table!

It is not necessary to make a distinction between the diverse test levels. But if this is desired nevertheless, all user stories in which, for example, the quality characteristic of 'functionality' is mentioned can be grouped into a system test, and all user stories with 'usability' and 'suitability', for example, can be bundled to form an acceptance test. And these can be executed during or parallel to the sprint or even afterwards if required.

The test strategy forms the basis of all (test) activities, processes and projects. It contains the legitimacy concerning what must be tested and how, which risks are inherent in the process, and which of these ought to be covered and how. This may influence the priorities in a scrum project, in the sense of the greater the risk, the higher the priority, for instance. Decisions involving what should or should not be done, with respect to time, costs and result (quality) may also be influenced. In a nutshell, a well-considered test strategy facilitates the entire scrum team.

Control

In scrum, the control phase of TMap is actually more of a facilitating activity than a restraining one. The team members must rely upon one another and, in turn, be trusted by the management. One component of management is the 'daily scrum', in which the progress of the test activities is reported and made transparent to all; any required adaptations to the priorities are discussed (see figure 30).

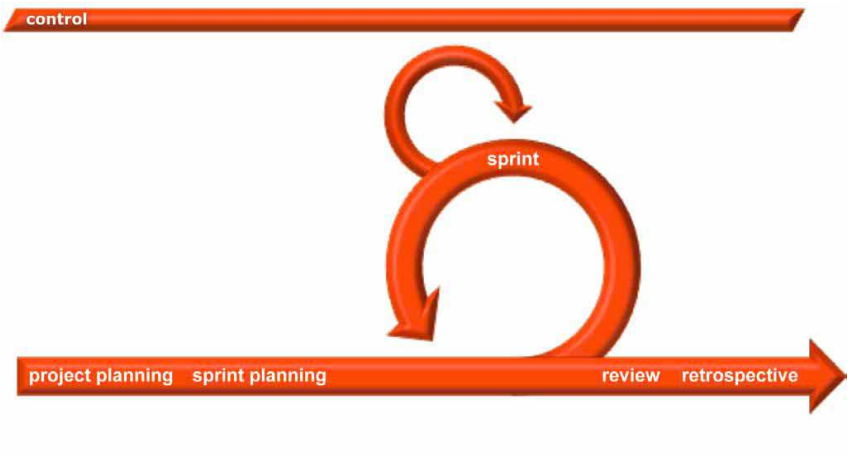


Figure 30. Control activities.

Progress

A pragmatic way to ensure that the progress is clear is to extend the test strategy table with a few columns, such as 'Tests created Y/N', 'Tests executed Y/N' and 'Tests passed Y/N'.

Item	Characteristic	Damage	Ch. of failure	Risk class	Intensity	Test design technique	Tests created (Y/N)	Tests executed (Y/N)	Tests passed (Y/N)
A	Functionality	3	3	9	•••	MCC			
	Usability	2	1	2	•	SYN			
B	Functionality	2	2	4	••	ET			
	Security	3	2	6	••	SEM-MCDC			
C	Functionality	2	1	2	•	DCoT-EQ			
D	Performance	2	1	2	•	EG			
E	Performance	1	1	1	•	EG			
F	Functionality	2	2	4	••	ECT-MCDC			
	Suitability	2	2	4	••	PCT-TDL2			
..	..	.	.	.					

Figure 31. Test progress table.

With this table (figure 31) on the whiteboard, everyone can understand the test progress at a glance. Or, if the rows of the test strategy table have been translated into tasks on the sprint backlog, again everyone will be able to see how much (test) progress has been achieved. Finally, the tasks will be listed in the current status column on the scrum board: to do, in progress, and done. Both the product and the sprint burndown charts also

provide information on the current status and progress, and adjustment may take place on this basis.

Defects
Any defects discovered are communicated during the sprint and the measures to be taken are discussed and implemented. Defects that can be rectified in a sprint are not registered. In such a case, the tester and the developer often collaborate to solve the defect in a rapid way. This facilitates the reduction of documentation and focus on progress. To safeguard against the rectification of defects becoming a main activity, the following guidelines are provided. A defect is recorded in the defects administration when:

- **the defect cannot be solved within one day**
- **the team decides — in consultation with the product owner — to rectify the defect in another sprint**
- **a defect discovered during the sprint review cannot be rectified in the sprint review.**

Of course, the team may deviate from such protocol if required. If the team believes that more than the minimum ought to be registered, that is fine. This may cover cases such as when metrics have to be built up, for example, but this should be recorded in the definition of done.

'Standard defect procedures' can be used for the identification and treatment of defects. However, there is one major difference. It is not the test manager but the scrum team that is responsible for the registration and monitoring of the defects. In the case that a defect surpasses the scope of the scrum team, this can be designated to the scrum master.

Setting up and maintaining infrastructure

As described previously, it is advisable to begin by setting up of the test infrastructure in a sprint 0 (see figure 32).

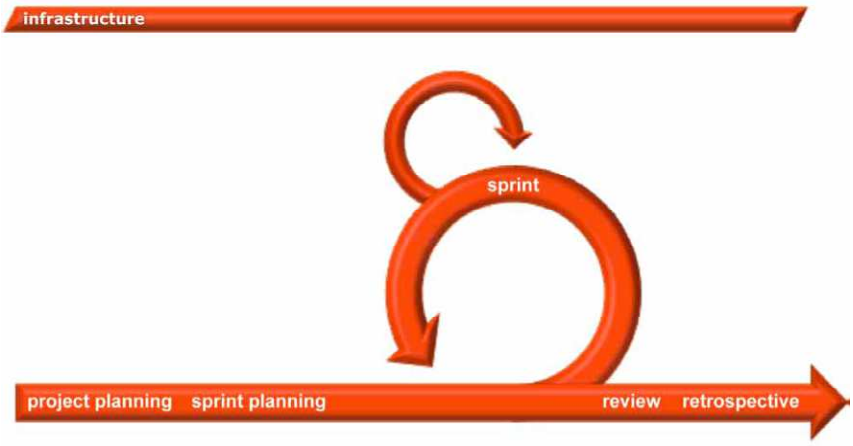


Figure 32. Setting up and maintaining infrastructure activities.

The test infrastructure consists of test environments, test data and test tools, among other things. The management of these takes place during the entire scrum project. The scrum team is often expected to set up and maintain the test infrastructure itself. This means that,

in this situation, there must be enough team members with sufficient knowledge to do this adequately.

In the situation where the test infrastructure occurs outside of the team, it is essential that response times are at a minimum. The test infrastructure must be stable, in view of the short duration of a sprint and the short period in which the test must take place.

Problems in the test infrastructure can have a great influence on the progress of the activities in the sprint.

Preparation

The evaluation of the test basis is an activity that is performed for each product backlog item and begins right at the outset of the project. Evaluating the product backlog items takes place in parallel to the development of other product backlog items (see figure 33).

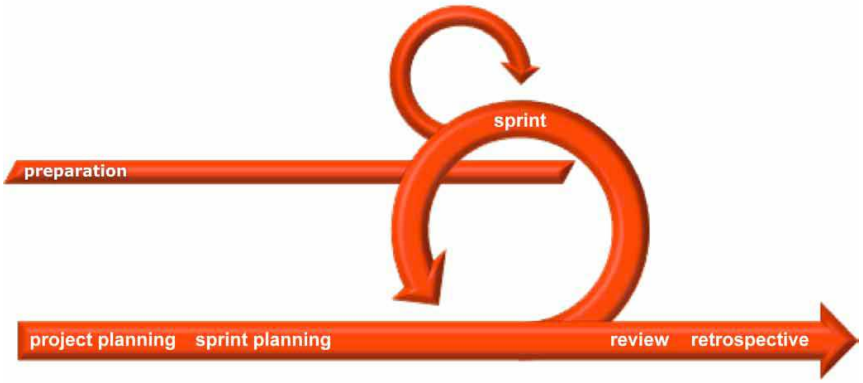


Figure 33. Preparation activities.

By requesting an explanation of every product backlog item from, or by posing critical questions to the product owner and the user (with knowledge of the subject matter) and the developer (with technical knowledge), the tester can obtain insight into the criteria that the product backlog item has to fulfill. Of course, this remains an interaction, because the critical questions enable the other team members to make qualitative improvements to their products. Not everything has to be documented; it is more important to communicate so that no information is lost.

When the tester has sufficient information to specify test cases, the evaluation — preparation phase — of the product backlog item has been completed. Writing a testability review is superfluous in this situation, because all interested parties have already been informed and any necessary measures have already been taken. If supplementary criteria have been included in the definition of done, these must be met.

If another team member — other than the tester — is allocated a test role, it may be useful to give this member a checklist on the basis of which the evaluation can be executed. This checklist contains questions about the completeness and consistency of the product backlog item, the degree to which the chosen test design technique can be applied, etc. This ought to guarantee the quality and uniformity of the evaluation.

On the basis of the evaluation results, the product risk analysis and the test strategy may have to be adjusted or a different test design technique may have to be chosen.

Specification

The specification of the test cases is an activity that is executed for every product backlog item. The specification takes place in the sprint (see figure 34).

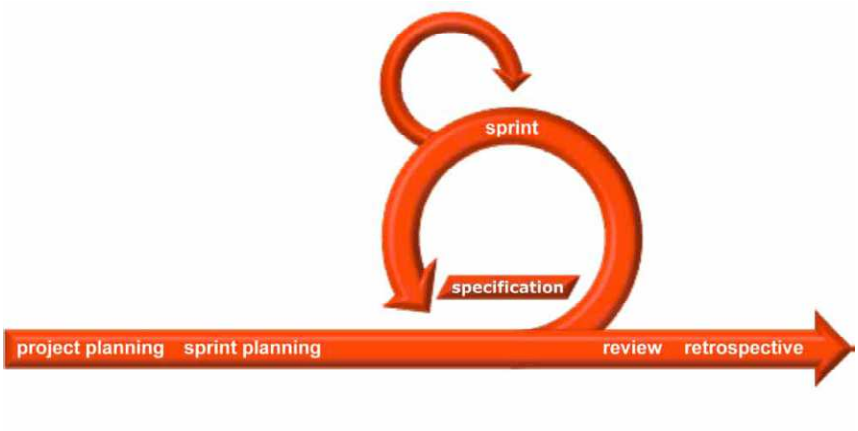


Figure 34. Specification activities.

In concrete terms this means that, if a test strategy table (figure 29) has been set up, the test cases for the relevant product backlog item are created according to the test design techniques assigned to that product backlog item.

The depth with which the test cases should be documented may depend on the demands made by the organization in connection with the transferability, repeatability and test automation requirements or the rules and regulations. In that case, this is documented in the definition of done. In all other cases, the documentation must have the degree of depth that enables the tester to execute the tests.

In view of the short lead time of a sprint, it is advisable to automate the tests immediately, or at least set them up in such a way that they can be automated. Unit tests are almost always executed in an automated way — this can be specified in a definition of done — and, in order to become familiar with both the technique and the product, it is advisable to involve the tester in this process.

When carrying out evaluations, the tester frequently receives a great deal of product information from the product owner, user and developer, which is processed immediately in the test cases, but is not always incorporated into the product backlog items themselves or other design products. In practice, it regularly turns out that the test cases contain more product information than the original product backlog items. As such, the test cases form a valuable source of knowledge, which can offer benefits in subsequent sprints and projects. One should ensure, certainly in this situation, that the preservation of testware is specified in the definition of done.

Execution

The execution of the test cases for a product backlog item is often done parallel to the test execution of other product backlog items (see figure 35).

Figure 35. Execution activities.

The execution of the pretest is generally limited in magnitude and sometimes not even necessary. After all, the tester is present in the team, watches the unit tests, and knows exactly what to expect. Due to the absence of transfer moments, to independent test teams for example, the execution of a pretest becomes superfluous.

During the execution of the tests, any defects are revealed.

Test-driven development

Scrum projects often use a test-driven development (TDD) approach. This is an approach for software development in which tests are written first, and only then is the code written. Although TDD is actually more of an eXtreme Programming approach than a scrum approach, it is nevertheless frequently used in scrum. A few advantages of TDD are: it is oriented to the perspective of the user. The test cases for which the code is written are based on the backlog items that have been formulated from the standpoint of the product owner. Due to the fact that all code is tested right from the start, this stimulates more trust on the party of the product owner.

Wherever there are advantages there are also disadvantages: in TDD, the programmer writes both the (unit) tests and the code for the application. This means that if the programmer overlooks something, this will be missing from both the test and in the code. A pitfall that occasionally occurs in real-life practice is the meager attention given to or even total absence of the performance of integration and system tests. These problems can be avoided with a good PRA and test strategy.

Test automation

As mentioned previously, it is advisable to execute the tests in an automated way. This certainly applies to unit and regression tests. But other tests can also benefit from automation, as a component of the continuous build and of integration strategies, for example. As is the case with many aspects, the actual plan to develop and execute an automated test ought to be included in the definition of done.

In practice, various solutions are chosen for the automation itself, sometimes during the sprint by the team members themselves, sometimes parallel to the sprint and by others. Another approach that is occasionally applied in practice is to automate test cases, created and manually executed during the sprint, parallel to the next sprint and to include them in a regression test. In the subsequent sprint, a regression test can be executed in an

automated way. This takes place parallel to the execution of the manual test cases of the corresponding sprint. And so on. At the end of the project — or perhaps just afterwards — a complete automated regression test will then be available. Here too, this must also be a part of the definition of done.

Completion

The evaluation of the test process dovetails perfectly with the past (sprint) retrospective, after which suggestions for improvement must be implemented in the following sprint as much as possible.

The preservation of the testware takes place during or at the end of the sprint or at the end of the project (see figure 36). Which and how much testware should be preserved, and whether or not a configuration management tool should be used, is specified in the definition of done.

A scrum project always consists of more than one sprint. Otherwise it is not worthwhile setting up a whole project. It is also important to pay attention to the construction of a regression test set. This can be done by, for example, including the most important test cases from the current sprint in a regression test set. Which of them are the most important can be determined with the aid of the risk table (see figure 28). The magnitude of this set increases with each sprint. And if a regression test must be executed in a following sprint, in addition to the testing of current items, there is often too little time to do everything. It is therefore advisable, prior to the formulation of a regression test, to reflect on how the regression test could be executed in an automated way.

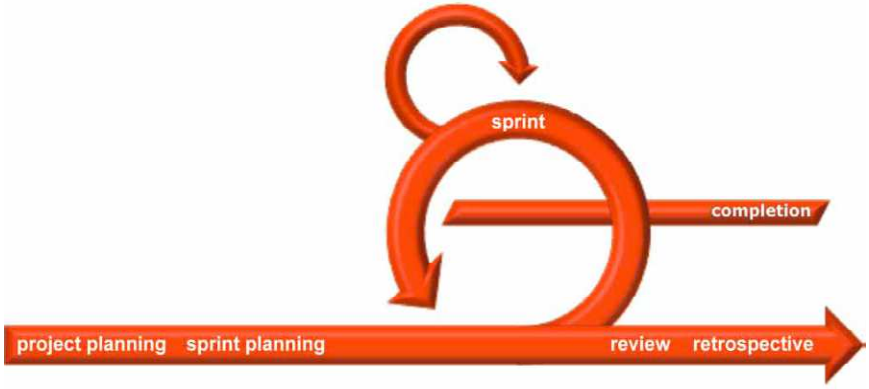


Figure 36. Completion activities.

'Responding to change' is an agile value that applies not only to changes in backlog items, but also to changes in roles and team set-ups. This response to changes, in test roles for example, can only be performed properly if sufficient attention is paid to maintainable, transferable and reusable testware. In this situation, testware management is an important aid. And here too, it is again essential that this be included in the definition of done.

4.1.5 Process: acceptance and system tests

The acceptance test and system test are considered as autonomous processes to be organized. They have their own test plan, their own budget, and often their own test environment to. They are processes running parallel to the development process, which must be started while the functional specifications are created. The TMap life cycle model

is used both in the creation of the test plan and in the execution of the other activities in the test process.

Life cycle model

Like a system development process, a test process consists of a number of different activities. A test life cycle model is necessary to structure the various activities and their mutual order and dependencies. The life cycle model is a generic model. It can be applied to all test levels and test types and used in parallel with the life cycle models for system development. In the TMap life cycle model, the test activities are divided across seven phases: Planning, Control, Setting up and maintaining infrastructure, Preparation, Specification, Execution and Completion (see figure 37 "TMap life cycle model"). Each phase is split up into a number of activities.
Using a test life cycle model enables the organization to keep an overview during the test process. By recording *what* has to be done *when*, *how*, *with what*, *where*, by *whom*, etc the claims to and the relationships with other aspects like techniques, infrastructure and organization are made automatically.

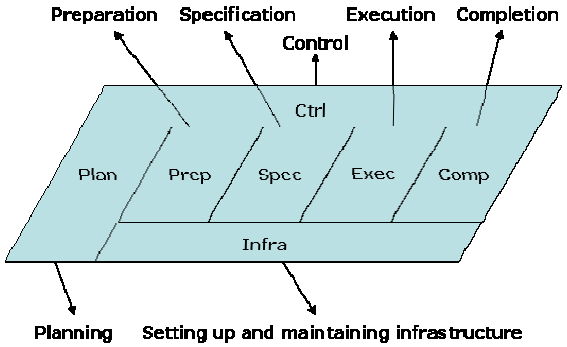


Figure 37. TMap life cycle model

The critical path and the shape of the life cycle model

If we were to compare the test process with an iceberg, only the Execution phase would be 'visible'. This means that only the Execution phase should be on the 'critical path' of a project. All activities in the other phases can be done either before or after.

The form of the life cycle model (parallelogram) shows that the test phases do not have to be executed strictly sequentially.

Test life cycle model relationships

The relationship between the TMap test life cycle and system development life cycle depends on the system development method used and the relevant test level. However, two 'fixed' relationships can be indicated. The start of the Preparation phase has a relationship with the moment at which the test basis becomes available; the start of the Execution phase has a relationship with the moment at which the test object becomes available.

Planning phase

The activities to be executed in the Planning phase create the basis for a manageable and high-quality test process. It is therefore important to start this phase as quickly as possible. The planning phase is an important test phase but is almost always underestimated. Often, the framework for a certain test level is are already defined at the

overall level in a master test plan. In this case, the detailed elaboration occurs in this phase.

After the test assignment has been finalized, an overall introduction to the test basis, subject matter and organization (of the project) is made. It is impossible to test the system completely. Most organizations do not have the time and money for that. This is why the test strategy, estimate and planning are determined according to a risk analysis process (BDTM steps 1 through 4), of course always in consultation with the client. It is then determined which test techniques must be used (BDTM step 5). The objective is to realize the best achievable coverage at the right place within the defined BDTM frameworks. The first steps in setting up the test organization and test infrastructure are also made. These activities are executed and laid down in the test plan for the relevant test level at the beginning of the test process.

Control phase

The primary test process is rarely executed according to plan. As such, the execution of the test plan also has to be monitored and adjusted, if necessary. This is done in the Control phase. The aim of the activities in this phase is to control and report on the test process in an optimal manner, such that the client has adequate insight into and control over the progress and quality of the test process and quality of the test object.

The test manager and/or administrator manage the test process, infrastructure and test products. Based on these data, the test manager analyses possible trends. He also ensures that he keeps well informed of the developments beyond testing, such as delays in development, upcoming big change proposals, and project adjustment. If necessary, the test manager proposes specific control measures to the client.

Information is the main product of testing. To this end, the test manager creates different kinds of reports for the various target groups, taking account of the BDTM aspects of result, risks, time and cost (BDTM step 6).

Setting up and maintaining infrastructure phase

The Setting up and maintaining infrastructure phase aims to care for the required test infrastructure and resources. A distinction is made between test environments, test tools and workplaces.

Setting up and maintaining the infrastructure represents a specific expertise. Testers generally have limited knowledge in this respect, but are highly dependent on it. No test can be executed without an infrastructure. All responsibilities in relation to setting up and maintaining infrastructure are therefore usually assigned to a separate management department. In a testing programme, therefore, the team will have to collaborate closely with these other parties that may be external to the organization. This means that test managers are in a situation in which they do not have control over the setup and maintenance of the infrastructure, but depend on it. This makes the setup and maintenance of the infrastructure an important area of concern for the test manager. It is a separate phase in the TMap life cycle model to maintain focus on it during the test. This phase runs in parallel to the Preparation, Specification, Execution and Completion phases. Dependencies with activities in other TMap test phases exist for some Setting up and maintaining infrastructure activities.

Preparation phase

The testability review of the test basis is done in the Preparation phase. The ultimate goal of this phase is to have access to a test basis of adequate quality to design the tests, which has been agreed with the client of the test.

Furthermore an early intake of the testability review of the test basis improves quality and prevents potential costly mistakes. This is because the development team works on developing the new information system on the basis of system documentation (which is part of the test basis). This documentation may contain errors that can cause a lot of – often costly – correction work if they are not detected in a timely manner. The earlier an error is found in a development process, the easier (and cheaper) it can be repaired.

Specification phase

The Specification phase specifies the required tests and starting situation(s). The aim is to prepare as much as possible so that tests can be executed as quickly as possible when the developers deliver the test object. This phase starts once testability review of the test basis is completed successfully. The test specification runs in parallel to, and in the shadow of, the realization of the software.

Execution phase

The aim of the Execution phase is to gain insight into the quality of the test object by executing the agreed tests.

The actual execution of the test starts when the test object, or a separately testable part of the test object, is delivered. The test object is first checked for completeness. It is then installed in the test environment to assess whether it functions as required. This is achieved by executing a first test, the so-called pretest. This is an overall test to examine whether the information system to be tested, in combination with the test infrastructure, has sufficient quality for extensive testing. The central starting point is prepared if this is the case. The test can be executed on the basis of the test scripts created in the Specification phase. In this case, the starting point must be prepared for the test scripts that are to be executed. The test results are verified during execution. The differences between the predicted and actual results are registered, often in the form of a defects report.

Completion phase

The structured test method of TMap can yield many benefits in the repeatability of the process. It allows products to be reused in subsequent tests if they comply with certain requirements. This may speed up certain activities. Products may be tangible things like test cases or test environments (testware), but also non-tangible things like experience (process evaluation).

When preserving the testware, a selection is made from the often large quantities of testware. Testware means, among other things, the test cases, test scripts and description of the test infrastructure. During the test process, an attempt was made to keep the test cases in line with the test basis and the developed system. If this was not (entirely) successful, the selected test cases are first updated in the Completion phase before the testware is preserved. The advantage of preserving testware this way is that it can be upgraded with limited effort when the system is changed to execute a (regression) test, for instance. There is therefore no need to design a completely new test.

Furthermore, the test process is evaluated in this phase. The aim is to learn from the experiences gained and to apply these lessons learned in a new test, if any. It also serves as input for the final report, which the test manager creates in the Control phase.

4.1.6 Process: development tests

Development testing is understood to mean testing using knowledge of the technical implementation of the system. This starts with testing the first/smallest parts of the system: routines, units, programs, modules, objects, etc. After it has been established that the most elementary parts of the system are of acceptable quality, the larger parts of the system are subjected to integral testing. The emphasis here is on data throughput and the interfacing between e.g. the units up to the subsystem level.

Place of development tests

The development tests are an integral part of the development work executed by the developer. They are not organized as an autonomous process for an independent team. Despite that, a number of different activities for the development test process, with their mutual order and dependencies, can be identified and described with the aid of the TMap life cycle model. The detailed elaboration may vary per project or organization and depends, among other things, on the development method used and the availability of certain quality measures.

An important quality measure is the concept of the agreed quality. To this end, the expectations of the client in relation to the craftsmanship and product quality must be made explicit during the planning to set up development testing. Examples of other quality measures are: test-driven development, pair programming, code review, continuous integration, and the application integrator approach.

Differences between development and system/acceptance tests

The development test requires its "own" approach that provides adequate elaboration of the differences between the development test and system/acceptance test as described below:

- As opposed to the system and acceptance tests, development tests cannot be organized as autonomous processes for more or less independent teams.
- Development testing uses knowledge of the technical implementation of the system, thereby detecting another type of defects than system and acceptance tests.
- In the development test, the person detecting the defects is often the same as the one who solves the defects.
- The perspective of development testing is that all detected defects are solved before the software is handed over.
- It is the first testing activity, which means that all defects are still in the product.
- Usually, the developers themselves execute development tests.

4.2 Test professionals

4.2.1 Introduction

A great variety of expertise is required for a tester to be able to function well in the discipline of testing. A tester needs to have knowledge of:

- The domain (e.g. logistical processes or financial reports)
- The infrastructure (test environment, development platform, test tools)
- Testing itself.

The management is responsible for ensuring that the right person with the right expertise has the right job, preferably in collaboration with personnel and training experts. A carefully controlled inflow and internal mobility policy supported by related training for test personnel are required. However, the negative image of testing makes suitable and experienced test personnel scarce.

The challenge for HRM lies in this combination of the negative image on the one hand and the importance of testing on the other, who can we find to execute this task and, more particularly, how can we keep them happy? An important tool to achieve such satisfaction is to offer the tester a career path.

This section discusses how to handle this issue. Below, "Points of concern" devotes attention to a number of points that require attention when setting up HRM for test professionals. The section on "Characteristics" describes what makes a tester a tester. What, for instance, are the personal characteristics of a tester? The next section ("Career path") gives insight into a possible career structure for testers, followed by a section ("Positions") describing the possible positions. Finally, the last section "Training" discusses the aspect of training.

4.2.2 Points of concern

Despite the fact that everyone is aware of the use and added value of testing these days, its image is not exemplary in every organization. Sometimes a test position is considered boring, mind-numbing and not very challenging. Or, it is perceived as the final stop in one's career or a necessary side-road when there is really nothing else to do. We describe a number of points of concern to set up HRM for test professionals below.

Tasks, authorizations and responsibilities

Many organizations have a comprehensive competency profile, the growth opportunities (in roles and salaries) and the available courses for roles and jobs. Such a profile is sometimes missing for test professionals, with the excuse that testing is a one-off activity for instance. It may be clear that this is not the case. This is why a written career structure is necessary for testers as well.

In more detail

Growth opportunities for testers
The job description of a tester must make clear the growth opportunities both within and outside the discipline of testing. Since testing acts at the crossroads of many professions, there is a range of (external) directions for growth. For instance, a tester who is regularly involved in testing a specific business application may evolve into a process analyst for that specific domain. Something that often happens as well is that an experienced test manager is asked to become a project manager.

Training options

Since testing is a risk-mitigating measure, a tester is a risk in and of himself. If the test professional does not test correctly or adequately, certain risks cannot be resolved. As such, it is important for a tester to know not only what well structured testing is, but also what he is testing. Taking the definition of a product risk in account (see also section 2.6 "Building Block 6: Product Risk Analysis"), the tester must have a feeling for the domain (damage part) and the technology (chance of failure part) on which the system is based. He must master them and know where the risks are generally (e.g. that one calculation or that one specific combination of architecture and hardware). Concretely, this means that testers, too, can (must) attend courses relating to e.g. the tool that is used for programming or the domain for which the solution is being built.

Workplace location

Because testing is at the crossroads of many professions, testers have a lot of contacts with the professionals in these disciplines. Putting the testers in the middle is killing two birds with

one stone. Their image is that they are truly at the crossroads, and there they have a lot of contacts. There are examples of an improved test process after the test team physically moved to the 'centre' of the organization. Among other things, it improved the mutual respect between programmers and testers, which in turn had a positive impact on quality.

Performance reviews

Performance reviews are generally performed by a superior with experience in the tasks executed by the person reviewed. This is the only way to achieve an objective picture of the past period and reach agreements. It is done this way in many IT disciplines. A programmer, for instance, is assessed by a project leader who used to work in programming himself. An information analyst is assessed by a business analyst with a similar past. Testers are often reviewed by the project leader. In his current role he has a lot to do with it, but he was never a tester himself. To avoid any suspicion of conflicting interests, a tester should be assessed by a superior or immediate stakeholder with actual testing experience. For instance the test coordinator or test manager.

Compensation

One present-day trend is to offer a variable salary in addition to a fixed salary. The size of the variable component (bonus) depends on the realization of certain objectives (Key Performance Indicators or KPIs). A tester in an organization has different interests than e.g. an information analyst, programmer or project leader. A test professional must be held accountable for other results. The situation in which everyone (including the tester) is held accountable for achieving the project planning is not ideal. The tester is at the end of the workflow and is often – incorrectly – perceived as the one causing the delays. It is better to assess a tester on the basis of his work's results. Examples are achieving his planning for one test cycle or the number of incidents during production. Clearly, a number of principles apply in this context. Never award a bonus to a tester for the number of defects detected, because this depends on the quality of the software (and is therefore someone else's point of concern).

4.2.3 Characteristics

What are a tester's characteristics, in other words, what properties must a person have to be an ideal tester? In the first place, the ideal tester does not exist. It varies per situation. We can, however, list a number of generic properties:

Communication, spoken and written

The tester maintains contacts with many different parties. For instance, he talks to e.g. the programmer, the information analyst, the project leader and other testers. It is important for a tester to be able to understand the interests of his discussion partners and communicate effectively. Written communication is important to record defects and write reports.

Accurate and analytical

A tester must focus on detail. It is important to establish for every requirement or wish what is actually being asked. In case of doubt, questions must be asked. It is important for the tester to go about his job analytically and refrain from making assumptions. A test basis is at the basis of his test, if this is not complete or contains defects, it is registered as a defect. A tester must never ever make assumptions in this respect, even though they may be self-evident.

Example

A test of a financial application required sums to be shown in Euros and dollars. The requirement contained a list of screens in which this occurred. Careful analysis by the tester showed that there were more screens in which this could occur. When the client was questioned, it was found that the requirement was indeed incomplete and the list of screens was modified. If the tester had not performed a full analysis, incomplete screens would have been taken into production.

Convincing and persevering

A tester communicates the detected defects to the party that caused them. This is where the extent to which the tester is convincing plays a part because the receiving party must consider the reported defects as actual defects. The tester must have power of conviction and persevere in affirming the importance of the quality of the product.

Objective and constructively critical

When a defect is communicated or questions are asked about a requirement, it is important to do so objectively. Comments like "bad software", "again an incorrect requirement" or "irritating colors" should not be used. In discussions about defects, it is important that the tester makes the problem clear to the other parties in a constructive, positive way. This means a certain level of diplomacy and refraining from pointing fingers at various parties.

Creative

The tester must simulate reality to make a statement about the quality of the software. Test cases are created, test data compiled, and a test environment defined for this purpose. This requires creativity.

Sensitive

The tester is at a crossroads between professions. The point of gravity of the tester's activities lies at the end of a process, when the pressure is highest. The tester must be aware of the tensions and interests and handle them correctly, so that the required objectives can be realized.

4.3 Acceptance and System Tests

4.3.1 Introduction

Acceptance test and system test

This chapter describes the TMap life cycle model, with the associated activities, for the test levels acceptance test and system test. Both can actually be considered (and therefore organized) as autonomous processes. They have their own test plan, their own budget, and often their own test environment to execute the test. They are processes running parallel to the development process, which must be started by preference while the functional specifications are being created.

A separation can be made in a development process between the client on the one hand and the supplier on the other. In the context of testing, the first group is summarized as the accepting (demanding) party and the second as the delivering party. Each of these parties has its own responsibility in testing. The supplier executes the system test to determine whether the system complies with the functional and technical specifications. This demonstrates that everything that needs to be delivered is actually being delivered. After the supplier has executed the system test, reworked the detected defects and subjected them to a retest with a positive result, the system is offered to the client for

acceptance. The accepting party wants to determine, with the test, whether what has been asked for is actually being delivered and whether it can do with the product what it wants to/must do.

TMap life cycle model

The process of the acceptance and system tests consists of a number of different activities. The TMap life cycle model is used to map the various activities, with their mutual order and dependencies. It is a generic model and can be applied for both test levels. However, the acceptance test and the system test each give their own interpretation to the life cycle model. In the TMap life cycle model the test activities are divided over seven phases (see figure 38 "TMap life cycle model"). These are the phases Planning, Control, Setting up and maintaining infrastructure, Preparation, Specification, Execution and Completion.

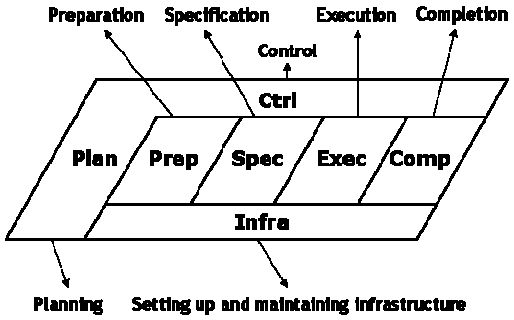


Figure 38. TMap life cycle model

In the Planning phase, the test manager formulates a coherent approach that is supported by the client to adequately execute the test assignment. This is laid down in the test plan. In the Control phase the activities in the test plan are executed, monitored, and adjusted if necessary. The Setting up and maintaining infrastructure phase aims to provide the required test infrastructure that is used in the various TMap phases and activities. The Preparation phase aims to have access to a test basis, agreed with the client of the test, of adequate quality to design the test cases. The tests are specified in the Specification phase and executed in the Execution phase. This provides insight into the quality of the test object. The test assignment is concluded in the Completion phase. This phase offers the opportunity to learn lessons from experiences gained in the project. Furthermore activities are executed to guarantee reuse of products.

The phases described above do not always have to be executed strictly sequentially. For instance, test cases for a part of the test may still be specified (Specification phase) while the test execution (Execution phase) has already begun for another part of the test. This is a situation that often occurs in projects in which there is phased delivery of software. We also recommend making preparations for the activities in the Completion phase as early as during the Specification phase. This phenomenon – where phases do not have to be executed sequentially - is expressed in the TMap life cycle model by the sloping lines between the phases. This results in the characteristic form of the model: the parallelogram.

In more detail

Retesting in the TMap life cycle model
The life cycle model also provides space for retesting. Retests occur when defects are detected while executing the test cases. If a retest must be prepared and executed, it

may be necessary to go through some phases of the TMap life cycle model again. Depending on the situation, this may be limited to the Execution phase, e.g. if only defects in the software are to be solved. If defects in the test basis must be solved, it may be necessary to (re)plan the retest completely (in particular in the case of a extensive rework action of the test basis). The phases Preparation, Specification and Execution must then all be gone through again.

When the life cycle model is related to the system development life cycle, a number of relationships come to light. Figure 39 "Relationship between TMap life cycle and system development life cycle " shows an example of these relationships.

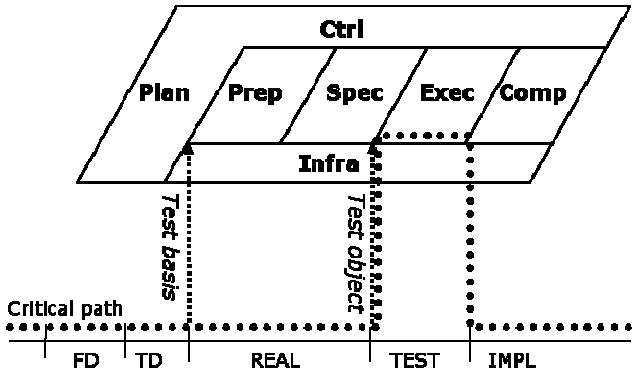


Figure 39. Relationship between TMap test phasing and system development phasing

The figure shows that the preparation phase of the TMap test life cycle can start once the test basis has been delivered. The test basis is created in the system development phases FD (functional design) and/or TD (technical design). After these system development phases, the realization of the test object begins (the system development phase REAL). The test (TEST) starts as soon as the test object is delivered. The next system development phase is the implementation phase (IMPL). This example demonstrates that only the TMap Execution phase is on the critical path of the project (the critical path is shown as a dotted line). All other test phases are executed in parallel to the other system development phases and, if ready in time, are not on the critical path.

In more detail

TMap life cycle model in relation to development models
The TMap life cycle model can be applied within various system development models. It does not matter whether system development occurs on the basis of principles such as waterfall, iterative or increments. The reason is that every system development model has the system development phasing as shown in figure 39 "Relationship between TMap life cycle and system development phasing". In iterative and incremental development (e.g. the RUP and DSDM methods), the first development phases in the model (FD, TD and REAL) must be seen as intermediary products. These are then tested (TEST) and integrated (INT). Figure 40 "Relationship TMap life cycle with increments" shows this schematically.

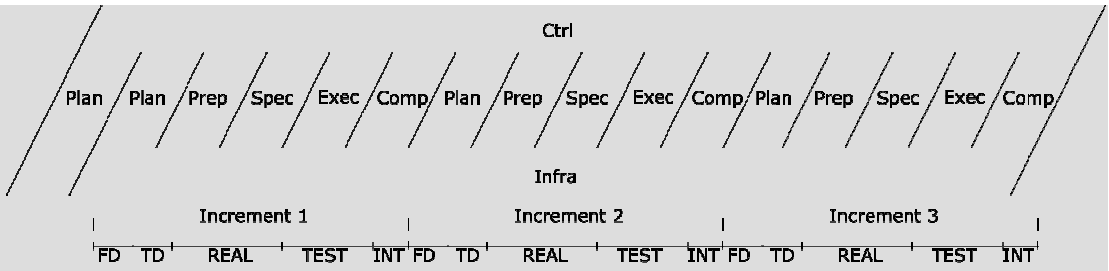


Figure 40. Relationship TMap life cycle with increments

At the project level, above all increments, the phases Planning, Control and Setting up and maintaining infrastructure are executed. The phases Planning, Preparation, Specification, Execution and Completion apply for every increment. The Planning phase in the increments is in close relationship to the master Control phase, hence the open link between the two. In view of the repetitive nature of iterative and incremental development, we must emphasize the repeatability of the tests. This can be achieved by e.g. the use of test tools and adequate testware management.

4.3.2 Planning phase

Aim

Formulating a cohesive and broadly supported approach with which the test assignment can be successfully executed. An important part of the planning phase is the creation of the test plan, for the purpose of informing the client and other stakeholders concerning the approach, schedule, budget, activities and the (end) products to be delivered in relation to the test process. If an overall master test plan exists, the test plan should be derived from it.

Context

All the steps of the planning phase should be gone through. The results are usually established in a separate test plan, if the test level is organized as a stand-alone activity. In some cases, particularly with iterative or agile development, the test level is integrated into the total process and the test plan is part of the project plan. The effort required to create the plan depends on what is already available. The presence of a master test plan, of Generic Test Agreements, or a Testing line organization with instructions, templates and standards can make creating the test plan significantly easier, as it is easy to refer to them. In creating the test plan, the test manager should allow for the possible and the impossible. An important factor here is the existing "testing maturity", or the quality of the test process. If there is familiarity with test phasing, if test tools are available and the testers are using test design techniques, how are the management and reporting normally managed? If the testing is not very mature, the test manager cannot expect too much from the test process or the testers involved in it. This applies to a lesser extent to the maturity of the development or maintenance process that surrounds testing. If this is chaotic and unmanageable, it is probably inadvisable to invest in the "perfect" test process; a "reasonable" test process will suffice.

Preconditions

To be able to make a meaningful start on the creation of the test plan, the following points should be known:

- The client for the test level
- Aim and importance of the system or package to the organization

- General requirements
- The organization of the development, maintenance or implementation process
- The (delivery) plan for the system to be developed or maintained, or package to be implemented
- The method of developing or maintaining the system or implementing the package
- If there is a master test plan, it should be fixed and approved
- Insight into the development and production environment, so that the test environment can be defined.

If this information is not yet available, for example because the development approach is still unknown, it will have a negative effect on the lead time, the effort required for the creation of the plan, or on the quality and required degree of detail.
Also required are the willingness and opportunity to agree on all kinds of aspects of the test process.

Method of operation

The test manager, as a rule, is the originator of the test plan. Ideally, a master test plan will be available. On this basis and in consultation with the client, he will formulate the assignment, making an allowance for the four BDTM [Business Driven Test Management] aspects of Result, Risks, Time and Costs (see section 3.1 "Business driven explained"). Subsequently, the test manager will prepare himself for the forthcoming phase by holding various discussions with stakeholders and consulting other sources of information, such as documentation. At the same time, he defines the assignment further in close co-operation with the client, and determines the scope of the test level.

In the event that, for the master test plan, a product risk analysis has not been executed, or if it is too general, a detailed analysis is carried out with the client and other stakeholders. This is done in order to establish the required results of the testing for the client (the *test goals*) and evaluate the risk level of the parts (*object parts*) and *characteristics* of the system or package to be tested. This analysis forms the basis of the test strategy and the process advances to an iterative stage. As part of the strategy based on the product risk analysis the tester determines the characteristics/object parts that should be tested, and with which test type and with which test intensity (the greater the risk, the greater the test intensity). Then the test costs are estimated in outline form and the test activities are planned (covering the biggest risks as early as possible). This is to be agreed upon by the client and other stakeholders and, depending on their views, possibly revised. In that case, the steps are then gone through again. In accordance with BDTM, the client therefore has a clear understanding of the test process and can manage the balance of Time and Money versus Result and Risk. Subsequent to this, the test manager refines the strategy further by determining test units and translating the decisions about test intensity into firm statements on which coverage is being aimed for. He then allocates *test approach(es), coverage types and/or test design techniques* to the characteristic/object part combinations, making allowance for the available test basis, resources and infrastructural provisions. Using these techniques, the test cases (and, for example, the checklists) are designed and executed at a later stage..

Figure 41 illustrates this.

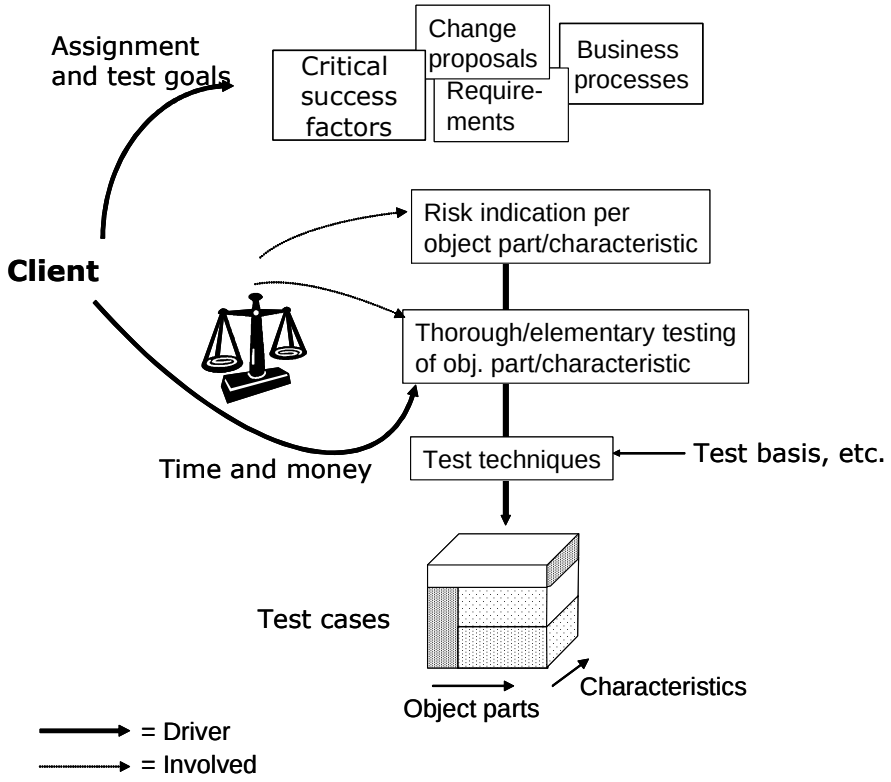


Figure 41. From assignment and test goals to test cases

Further steps in the plan formulation are that the test manager establishes the test basis, defines the test products and builds up the test organization. The test manager also defines the required infrastructure. Test management is furnished with procedures and standards, supported as far as possible with tools. As a rule the elements available in the master test plan, Generic Test Agreements, the test policy or the Testing line organization are used.
The most important risks that threaten the test process are cited, and possible measures are proposed for managing these risks. As a last step, the test manager has the test plan approved by the client. While the activities in this subprocess are described in sequence, in practice, certain activities will be done several times and/or in a different order. If, for example, certain infrastructure parts are required for a test and cannot be supplied, then the strategy may have to be adjusted.

Roles/responsibilities

The primary responsible role in the creation of the test plan is taken by the test manager, sometimes known as the test coordinator.

In more detail

Test manager or test coordinator?
While in this section the term of test manager is consistently used to refer to the individual responsible for the test process, in practice it is also often a test coordinator who heads the system or acceptance test. The differences are more emotional and circumstantial than objective, but generally, the following is the case:

- The more authorizations involved, the more the term of test manager is preferred

- The greater the scope of the test, ditto
- The greater the size of the test, ditto
- If an overall test manager is managing the overall test process, test coordinator is preferred
- If a test coordinator is coordinating the overall test process, test manager is preferred

Activities

The creation of the test plan involves the following activities:

1. Establishing the assignment
2. Understanding the assignment
3. Determining the test basis
4. Analyzing the product risks
5. Determining the test strategy
6. Estimating the effort
7. Determining the planning
8. Allocating test units and test techniques
9. Defining the test products
10. Defining the organization
11. Defining the infrastructure
12. Organizing the management
13. Determining the test project risks and countermeasures
14. Feedback and consolidation of the plan

The scheme below (figure 42) shows the sequence and the dependencies between the various activities. Every one of the activities may be gone through several times, as the result of an activity may mean a previous activity needs to be revised. As earlier indicated in the method of operation, the steps 5, 6 and 7 have an explicitly iterative character:

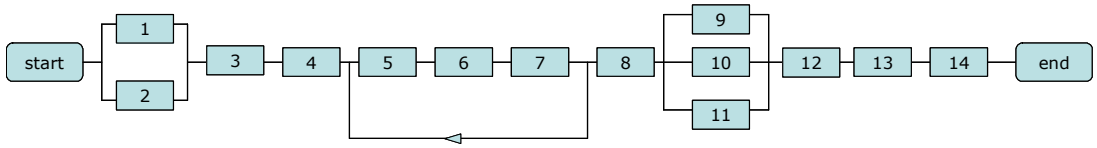
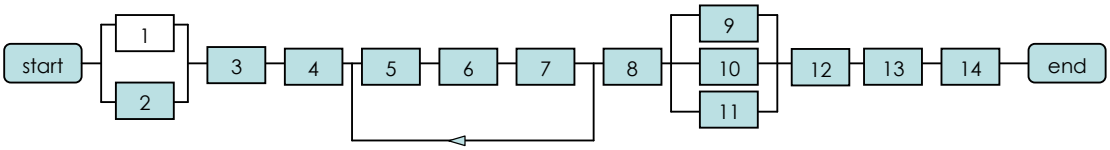


Figure 42. Creating the test plan

4.3.2.1 Establishing the assignment



Aim

A system test or acceptance test starts with the formulation of the test assignment so that the aim, tasks and responsibilities of the test level are made clear to everyone involved.

Method of operation

By establishing the assignment in the test plan, it is made clear to all the parties involved (including the client) what the test process is meant to deliver, and expectations are brought into line. The assignment for the test level should be compatible with the assignment as set out in the master test plan.

An assignment for a test plan consists of the following elements:

- Client
- Contractor
- Assignment
- Scope
- Preconditions and assumptions.

These parts are explained below:

Client

The party who has commissioned the creation of the test plan and the execution of the tests. It is important for the test level to acknowledge who has commissioned the execution of the test.

In more detail

In practice, we generally see the following possibilities for the various test levels:

System test	- Project manager from the supplier - Project manager /project leader for realization
Functional acceptance test	- Project manager from client/acceptors - Head of functional management
System integration test	- Project manager from client /acceptors - Head of functional management
User acceptance test	- Project manager from client /acceptors - Head of users organization
Production acceptance test	- Project manager from client /acceptors - Head of system management

Contractor

Usually, a test manager or test coordinator is responsible for creating the test plan and executing the test assignment.

Assignment

The assignment should be set up in consultation with the client and should indicate the aims and the scope of the testing.

In more detail

This would appear to be the obvious core of the activity – "Establishing the assignment". Despite the importance, in practice the formulation of the assignment is often somewhat abstract and generic, in terms of "providing a quality assessment" or "providing insight into risks". It is mainly in the scope, preconditions and assumptions (and later the strategy) that the total assignment is sharply defined.

In more detail

Iterations

Iterative or agile system development delivers a large number of (interim) releases or prototypes for testing. It should be clear from the formulation of the assignment that such an interim release or prototype may not be assessed on every aspect of a forthcoming

production system, but only on those aspects that are relevant to the interim release or prototype itself.

As test manager, you should ideally gain a feel for what the guiding principle of the project is in terms of BDTM. Is the client mainly concerned with Time or Costs, or is Result/Risk the driving force? This is no easy task, for the initial reaction ("our maximum budget is € …", and "the deadline of … is set in concrete") often seems crystal clear, but on further questioning is not always so ("… and if the system then only has ¾ of the functionality?"). Nevertheless, this insight will aid the test manager's understanding and facilitate later communication on the choices to be made. The sensitivity of this information means that it is not necessarily established in the plan.

Additionally, the test regularly involves secondary requests. The client should allocate budget and/or time available for these. Examples are:

- The creation of a *standard maintenance* test plan, to include all the reusable test aspects
- Training and coaching of the employees in testing
- Improvement and structuring of the test method of operation employed
- Implementation of a test tool
- The setup, use and maintenance of a scalable regression test set
- Supply of (automated) testware for the testing of subsequent releases.

In more detail

Usually, the client makes resources (people and means) available, or pays for them, for example, by hiring in people internally or externally. Payment usually takes place based on the number of hours. In certain cases, particularly in the case of outsourcing, when the testing is done by an external supplier, more creative agreements can be made. Below are some possible constructions that appear in practice:

- Fixed-price
  The supplier carries out the testing for a previously agreed fixed price. This usually includes a fixed number of retests. In the event of a breakdown in the test process owing to the client being unable to meet the set agreements, or if more (re)tests are necessary than were agreed, additional charges are applied. In the other cases, the risk is borne by the supplier.
- Fixed-price per test case
  A variation on the above is that a fixed sum is agreed per test case to be specified and executed.
- Fixed-date
  Similar to fixed-price, but with a fixed date of completion
- Fixed-date, fixed-price
  As above, with both a fixed price and a fixed date of completion
- Bonus-malus
  In addition to the above, agreements can be made with the intent of distributing the risk more satisfactorily among both parties. By doing this, the client pays the supplier by the hour with the understanding that there is a fixed date or fixed price. If the supplier requires fewer hours or less lead time, he is given a bonus in the form of more money. And an example of malus: if after X amount of time after going into production, critical faults arise, or if the timeline or hours are exceeded, the supplier gives a discount on the fees.
- Result sharing
  An unconventional form is when the supplier is paid with a percentage of the profits from the new system. In this case, the system is an investment for both the

client and the supplier, and both have every interest in a successful outcome. It will be obvious that this involves big risks (but also opportunities).

Scope of operation

The limits of the test operation should be indicated here. This should preferably be more specific than what is already stipulated in the master test plan. The following matters should be taken into consideration (where applicable):

- System(s)
- Conversions
- Administrative organization (AO) procedures
- Quality characteristics (allocated in the master test plan)
- Interfaces with adjacent systems (is the interface being tested up to the other system or *up to and including*, or even to include the entire chain?).

In respect to changes, it is important to determine the parts of the above that are being considered.
It is also important to indicate the issues that are outside of the scope of the testing. Besides those mentioned above, the following should be kept in mind:

- System changes that are not included in the project
- Test activities that are carried out by other test levels or parties
- Reorganizations
- Possible future projects that influence the current project (particularly if there is a lack of clarity concerning other projects).

Preconditions

Preconditions describe conditions set by third parties, such as the client, the project, managers or users with regard to the test process and within which the test process must operate.

For example

- Master test plan
  The master test plan drives the setup and execution of the test level
- Milestones
  Often, as soon as the test assignment is issued, a number of milestones are established, such as the delivery date of the test basis, the test object, infrastructure and the date of going into production
- Available resources
  The client often sets limits to the available people, resources and budget
- Norms and standards to be maintained
  From within the (test) organization or the master test plan, certain requirements may be set as regards method of operation, procedures, techniques, templates, etc.

Assumptions

Assumptions are external circumstances or events that must come about in order for the test process to succeed, but that are beyond the control of the test process. In other words, the requirements that the test process sets other parties.

For example

- Quality of preceding tests
  The preceding tests, e.g. development or system tests are carried out in the agreed manner

- Quality of test object
  The test object has the agreed entry quality. This should be established with the aid of so-called entry criteria, which overlap with (but are not necessarily the same as) the exit criteria of the preceding test
- Support to be supplied
  Within the test process there is a need for various forms of support, e.g. in respect of the test basis, test object, domain knowledge and/or infrastructure. This support may be required to a certain degree and/or for a certain period. Bear in mind, for example, the availability of developers for solving obstructive defects during the test execution. Usually, each test level has its own expertise. For instance, the users acceptance test will have little need of domain knowledge support, while the support needs will concern precisely the other types of expertise, such as technical or test-method support
- Changes in test basis and test object
  The test team should be involved in the implementation of changes. In most cases, this simply means following up the existing procedures within the system development process. For example, the test manager should participate in the Change Control Board in order to estimate the consequences of a change from the test point of view
- Delivery of the test object
  The development team delivers the test object in a number of different but efficiently testable parts, and takes responsibility for installation in the test environment
- Response time to defects
  How quickly should the project react to the finding of defects. Below is an example of such agreements:

Severity	**Priority**	**Response time**	**Lead time**
Test-obstructive	High	1 hour	4 hour
Severe defect	High	1 hour	1 working day
Regular defect	High	1 working day	2 working days
Regular defect	Low	1 working day	To be determined per defect
Cosmetic defect	Low	2 working days	To be determined per defect

The test manager cannot make do with including these points in the plan and then assuming when the plan is accepted that all the points have been organized. On the contrary, he should first agree the points with the parties that own them, so that the points constitute set agreements and not surprises. It is advisable to mention in the plan, per assumption, for which stakeholder or parties these are intended.

For a checklist of possible preconditions and assumptions, please refer to www.tmap.net.

Products

The assignment, established in the test plan.

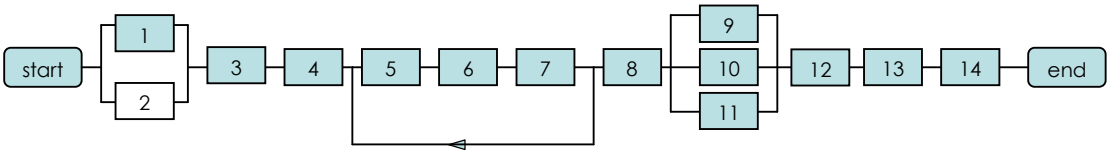
Techniques

Checklist of preconditions and assumptions (www.tmap.net).

Tools

Not applicable.

4.3.2.2 Understanding the assignment



Aim

To obtain insight into the (project) organization, the aim and purpose of the system development process, the system or package to be tested and the requirements to be met, so that better direction can be given to the other steps in the planning.

Method of operation

The method of operation covers the following activities:
1. Identifying acceptors, using acceptance criteria and other information providers
2. Examining the available documentation
3. Conducting interviews

In practice, this activity is carried out in parallel with the formulation of the assignment. It is also somewhat underestimated. Specifically, the test manager may speak to too few stakeholders, although it is essential in the beginning to measure expectations adequately and, as test manager, to 'put the feelers out' in all directions. This is necessary in order to be able to carry out the following activities effectively and to manage the test process successfully in the future.

1) Identifying acceptors, using acceptance criteria and other information providers

Usually the client is not the only stakeholder who has to accept the system; there are generally others, and it is important to clarify who these accepting parties are. This is done in consultation with the client. In practice, the test manager gets an opportunity here to discuss with stakeholders at a high level in the organization (steering group members) and to interpret their opinions and expectations. Often there is no other opportunity for this, unless the test manager is in the (unfortunately) rare position of regularly participating in the steering group discussions. It is important to establish which acceptors are to be provided with information directly or indirectly during the project by means of test reports. It should also be clear what requirements or acceptance criteria each acceptor is proposing. These are the minimum qualitative requirements that the product must meet to make it satisfactory to the acceptor. For the sake of clarity: the gathering of acceptance criteria is not the responsibility of the testers, but it is input into the setup of the test process. Acceptance criteria can be very diverse. Some examples are:

- Qualitative criteria as regards product and generation process, e.g. the number of defects that may remain open
- Criteria as regards the environment, e.g. the infrastructure should be installed or the users should have followed a training course
- Criteria in the form of (the detailing of) requirements of the product, e.g. 'an order should be processed within X seconds'.

Not all the acceptance criteria are relevant to testing. The first example has a considerable overlap with the exit criteria for the test process. The second example is usually less important to testing, and the third example is a form of test basis.

In more detail

Acceptance criteria pitfall
This latter use of acceptance criteria contains a danger. In practice, the following sometimes happens: after establishing and freezing the requirements, users discover that they have additional requirements. They then formulate these requirements as acceptance criteria. In this way, acceptance criteria form the 'back door' for taking in even more requirements. This is not a good method of operation. The only correct way is to submit a change proposal to a Change Control Board.

Besides acceptor, various other parties/individuals can supply the test process with relevant information. Bear in mind, for example:

In more detail

- The overall test manager, at coordinating level, for obtaining insight into the test assignment and what is expected of the test or the test manager
- The (representatives of the) client, for obtaining insight into the business aims and the 'culture' as well as the aims and strategic importance of the system
- The project manager or quality management employee, for obtaining insight into the steps and components of the development process and the correlations, with special focus on the (expected) place of testing in this
- The domain experts from the user organization, for obtaining insight into the (required) functionality of the system
- The designers, for obtaining insight into the system functionality to be developed
- System administrators, for obtaining insight into the (future) production environment of the information system
- Testers, for obtaining insight into the test method of operation and test maturity of the organization
- The suppliers of the test basis, the test object and the infrastructure, for guaranteeing coordination at an early stage among the various stakeholders.

2) Examining the available documentation

The documentation provided by the client is examined. For example:

In more detail

- Test documentation, such as the master test plan or a Generic Test Agreements document
- System documentation, such as stakeholder analyses, business or user requirements, or an information analysis, system requirements, functional and technical design
- Project documentation, such as the plan of approach for the system development process, organizational charts and responsibilities, the quality plan, review reports and a function-point analysis
- A description of the system development method, including the norms and standards
- A description of the test method applied, including the norms and standards
- Evaluations and points of learning from previous tests that may be relevant to the forthcoming test
- Contracts with suppliers

If the system development process relates to maintenance, then the availability and usability of existing testware is also investigated.

3) Conducting interviews

The various parties involved in the system development process are interviewed. An interview checklist is available at www.tmap.net.

The test manager asks the stakeholders questions concerning, besides the general background of the system to be tested and the process to be followed:

- Their expectations about the results of the testing – what do they hope to see as the end result? This may relate to business processes supported by IT, realized user requirements or use cases, change proposals, critical success factors, cited risks (to be covered) but also, for example, that the new system should have at least exactly the same functionality as the old system (therefore no regression). These are referred to as the test goals. Do they fit with the test goals of the master test plan?

Definition
A test goal is a goal that, for the client, is relevant for testing, often formulated in terms of business processes supported by IT, realized user requirements or use cases, critical success factors, change proposals or cited risks to be covered.

- Does the interviewee have an idea of what the characteristics are (usually the quality characteristics) and object parts that operate in the above? Some people are able to answer this very well, but it may be too complex for others. The test manager has to estimate how much detail the discussion will allow
- What is the test basis, if any, that may or should be used later on as proof that the test was thorough enough?
- What is the risk estimate of the test goals and/or characteristics/object parts? A risk is defined here as the product of Damage x (Chance of defect x Frequency of use). Often the interviewee will only be able to mention particular aspects of a risk, such as the Damage, the Frequency of use or the Chance of a defect. That doesn't matter, for it can be expanded upon in the next step, the product risk analysis
- Does the stakeholder wish to be reported to, and if so, at what level?
  A point of focus here is that the number of types of reports should remain somewhat restricted for practical reasons. If necessary, the test manager should discuss this with the client.

It is also advisable where possible to consult those indirectly involved. For example, the EDP auditors, the implementation manager, the future maintenance organization, etc.

Tip
Instead of individual interviews, a kick-off session could be organized with (a number of) the relevant parties. The advantage of this is that the various viewpoints help to arrive collectively at a clear picture. This often happens in particular when consultation is held to determine the (test) impact of change proposals.

The test manager feeds back the findings of this activity to the client for verification.

Products

This activity delivers the following parts of the test plan:

- Stakeholders and acceptance criteria
  The stakeholders relevant to the testing and their acceptance criteria
- Norms and standards

The standards employed are cited here. As regards testing, these can involve instructions issuing from the Testing line organization, the master test plan or generic test agreements, TMap, TPI or test manuals. Development standards, document standards or quality norms that have to be or will be followed are also possibilities

- Basis of the test plan
  Here the documents are mentioned that form the basis of this test plan. For example, a master test plan, project plan, specific project or test plans, a specific or a generic test method, generic test agreements, an implementation plan or other documents that are of importance.

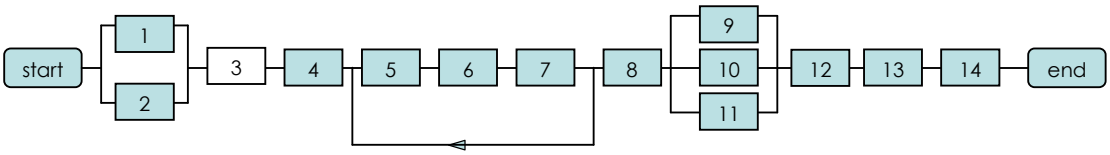
Techniques

Checklist 'Understanding the assignment' (www.tmap.net).

Tools

Not applicable.

4.3.2.3 Determining the test basis



Aim

The unambiguous defining of the test basis, so that it is known at an early stage what the test object is to be compared against.

Method of operation

The method of operation covers the following subactivities:

1. Defining the test basis
2. Identifying the test basis

1) Defining the test basis

The test basis, or the gathering of all the written and unwritten requirements with which the test object should comply, can take various forms. Bear in mind, for example, requirements, acceptance criteria, functional designs, technical designs, user manuals, interviews, reports of meetings, legislation, but not forgetting the old system, a previous release of the system, a prototype or even a domain expert. The gathering of non-documented test basis in particular is difficult; further information on this is given in section 6.5 "Preparation Phase". It is important when determining the test basis to ensure that the non-functional requirements are also known, such as e.g. requirements in respect of performance or security.

In more detail

The individual test levels often make use of various sources for obtaining the product requirements, against which testing is carried out in the test level. For example, the acceptance test is often focused on requirements that are described at the level of, let's say, business processes, whereas in the unit testing it is checked whether the technical requirements pertaining to a specific unit have been met. The test basis will therefore not be the same for all the test levels.

Something to be kept in mind in respect of the test basis is that these requirements should be as concrete and measurable (testable) as possible to prevent misunderstandings. This is often not the case in practice. Where possible, it can already be observed at this stage, otherwise it will become apparent at the later stages of Preparation and Specification. It is also possible that, at a later stage, it will be discovered that a requirement is very difficult to test. In such cases, it is agreed with the client whether a simplified test is acceptable.

2) Identifying the test basis

Establish, as far as possible, the identification of the relevant test basis. Bear in mind the delivery date, version, status, etc.

Products

The test basis to be used, established in the test plan.

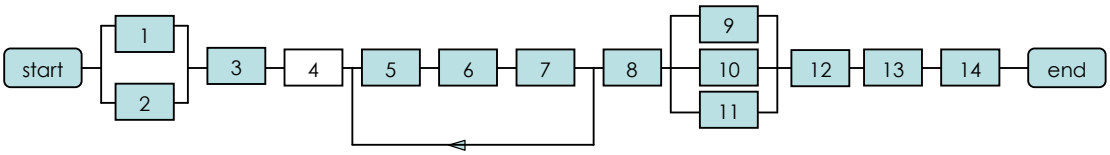
Techniques

Not applicable.

Tools

Not applicable.

4.3.2.4 Analyzing the product risks



Aim

To have the participants and the test manager arrive at a common perspective about the parts and characteristics of the system based on the risk level.

Method of operation

Testing is a measure for obtaining insight into the quality and related product risks of a system or package when it is put into production by an organization. Since time and resources are usually limited, it is important to determine the system parts or features that will require extra or less test effort early in the process. Practical choices must be made here. Performing a product risk analysis (PRA) will help determine the areas of focus for the test.

A PRA within the framework of a test level is optional: if the master test plan PRA has already been done in sufficient detail (i.e. at the level of characteristics and object parts), this step can be skipped. If there is no master test plan PRA, it must first be determined what the test goals are and what relationship they have with the characteristics/object parts that are to be tested in the relevant test level. This is done in a similar way to the PRA for the master test plan. If there is a master test plan PRA, but not at the level of object parts, then the PRA should be further refined.

The execution of a product risk analysis is divided into the following subactivities:
1. Determining the participants
2. Determining the PRA approach
3. Preparing sessions/interviews
4. Collecting and analyzing product risks
5. Checking for completeness

In section 2.6 "Building Block 6: Product risk analysis", these steps are explained in detail.

Tips

- A point to note is that the PRA session, if relevant, may be combined with (a part of) the subsequent activity, 'Determining the test strategy'.
- With a PRA for a test level, the challenge is to allow the test goals, characteristics and object parts to relate only to the scope of the test level. It is meaningless to recognize security as a big risk if this characteristic has already been assigned to another test level from within the master test plan.

Products

- The risk tables with test goals and possible object parts per characteristic with risk indications, managed separately and optionally established in the test plan
- The PRA overview of characteristics/object parts with risk class, established in the test plan.

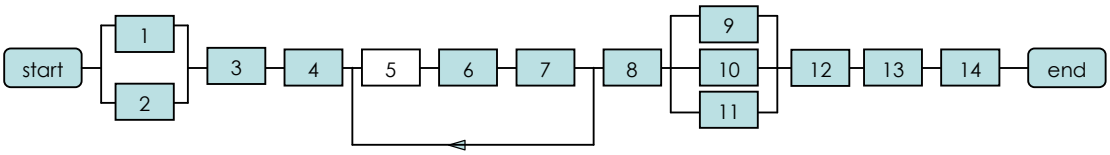
Techniques

Product risk analysis (section 2.6);
Explanation of quality characteristics (section 4.4).

Tools

Not applicable.

4.3.2.5 Determining the test strategy



Aim

To decide, based on the insight into the risk levels associated with the object parts/characteristics of the system, on the test types to be used, and on the test intensity for each (combination of) characteristic/object part of the system.

Method of operation

In defining the strategy for a test level, the choice is made about the test types and thoroughness of testing, i.e. the extent to which the combinations of characteristics and object parts are tested. This is dependent on the risk estimate from the PRA, or rather the degree to which the client wishes to cover these risks, and how much time/money he allocates for it.

To this end, the test manager makes a proposal for each combination of object part/characteristic in respect of the required test types and test intensities.

Test types
In this, the test manager specifies what is to be tested out of a particular object part-combination. At its simplest, this is the test of a quality characteristic, e.g. a functionality test or performance test, but often it is possible, and necessary, to provide more insight. Other test types are associated with the quality characteristic functionality, e.g. the multi-user test, regression test or chain test. An overview of "Applied test types" is included at www.tmap.net.

In more detail

Traditionally, most attention is given to testing functionality. More and more other characteristics, like suitability, security, portability, performance and usability, are being tested. Specifically the Internet has made these characteristics of systems more important and full of risks. The characteristics can be tested using test design techniques or checklist, just like functionality. Besides that, other points of attention and ways to test are available.

Test intensity
In determining the test intensity, a choice is made from the following possibilities:

•••	Thorough explicit testing
••	Average explicit testing
•	Light explicit testing
E	Evaluation
I	Implicit testing Testing in conjunction with another test type without making explicit test cases; only observable defects are documented.

– If a cell is left empty, this means that this particular evaluation or test level can ignore the characteristic.

In more detail

The result of this step is described below:

Example of ST

Characteristic	RC MTP	ST MTP	Subsys1	Subsys2	Total sys
Functionality	n/a	n/a	A/••• functional, regression	B/•• functional, regression	C/• integration, multi-user
Performance online	B	•	-	-	C/• random sample in ST environment
…					

RC MTP = Risk class assigned to the characteristic from within the master test plan
ST MTP = Test intensity assigned to the test (in this case, the system test) from within the master test plan
n/a = not applicable, in the master test plan risk class and test intensity are not assigned to the characteristic, but to the combination object part/characteristic
A = High risk class
B = Average risk class
C = Low risk class
The risk classes are taken from the PRA of the master test plan or (in more detail) from the PRA of the test level itself.

The test manager then supplements this table with the necessary explanations. In the above example, a functional test is to be carried out for subsystems 1 and 2 in respect of the new and changed functionality, and a regression test in respect of the unchanged parts will be furnished. Following the separate testing of subsystems 1 and 2, the total system will be tested as regards integration aspects; a multi-user test will also take place. Performance will be tested in the non-representative ST environment for a limited number of situations.

UAT example

Characteristic	RC MTP	UAT MTP	Subsys1	Subsys2	Total sys
Functionality	n/a	n/a	A/•• functional	B/-	C/• regression
User-friendliness	B	••	C/I	-	B/•• usability
Security	A	•			
- authorization matrix	B		-/E authorization test	-/E authorization test	B/•• process test
- application	C		-	-	C/• penetration test
Suitability	B	•••	B/• scenario test	C/• scenario test	A/••• process test

…					

In the above example, for subsystem 1 a functional test is performed again, using a number of the test cases from the ST, but in the AT environment. If several deliveries are made, a regression test on the total system takes place. A usability test in the own environment is carried out and in other tests an implicit test of user-friendliness is carried out. The authorization matrix is evaluated for correct content for subsystems 1 and 2. Also, the business processes, or subprocesses, related to subsystems 1 and 2 are simulated by means of running user scenarios. Subsequently, the operation of the total system is tested in combination with the business processes. A light penetration test is also planned.

An initial setup of the strategy is often possible in the PRA session or the PRA, and these steps of the Strategy definition can be combined. If this doesn't work, then the test manager makes a proposal.

A point to note is that when the MTP indicates a thoroughness of ••• for a particular test level (e.g. ST) or a particular combination of characteristic/object part, this doesn't mean that, in the ST, the entire system, or the combination of characteristic/object part, should be tested in the greatest possible coverage, but that testing is required with greater test intensity than average. This should also be evident from the MTP notes.

Tips

- With iterative or agile system development, the test strategy should focus on regression tests in respect of the many interim releases (iterations, increments). Also, the test (strategy) should be restricted to the characteristics of the interim release and not formulate a strategy as if it concerns the final release. This appears easier than it is, for in the PRA the users provide their risk estimate on the basis of the expected final release.
- At the setting up of the strategy, the test manager should make as much allowance as possible for the consideration of costs, time and skills required. If he knows that there is only a very limited budget or that the available people have no experience in testing, he should avoid proposing 'impossible' strategies, such as very expensive tests or very thorough test-design techniques (to prevent too may feedback cycles).
- It is advisable, when choosing between thorough and light testing of a characteristic/object part, to make an immediate inventory of the test basis to be used, as the availability and degree of detail of the test basis can influence the budget and planning. Allowance can be made for this during these steps.

In more detail

Maintenance
The chance of defects is the principal difference between new-build and maintenance. The formulation of the changes as object part facilitates the strategy. Several variations are possible:

- A limited test, aimed only at the change
- A complete (re)testing of the function in which the change was made
- The testing of the correlation between the changed function and the functions directly surrounding it
- A test of the entire system.

Regression test

The regression test of the system as a whole is recognized as well. It focuses on the coherence between the changed and unchanged parts of the system, since this is where the chances of regression are the greatest. If the PRA for the new-build is available, the risk categories applied here to the characteristics/object parts can play a role in the composition of this regression test. A regression test may be carried out to a full or limited extent, depending on the risks and on the required test effort. It is very easy, with the aid of the scalable regression test set (see section 6.6 "Specification Phase") to perform either a thorough or a light regression test. This makes for flexibility in the testing of later releases.

Products

The test strategy, established in the test plan, with a brief description of the planned test types and an indication of the importance per characteristic/object part.

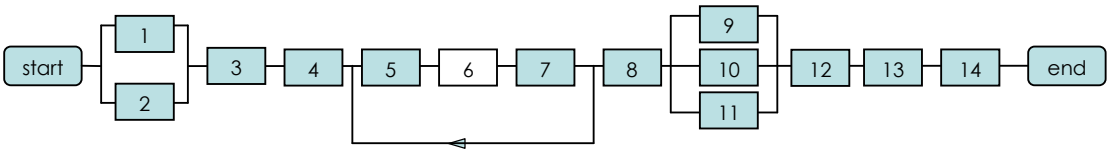
Techniques

Strategy determination (as described in this section).

Tools

Not applicable.

4.3.2.6 Estimating the effort



Aim

The estimation of the effort for the test level based on the test strategy, so that the client can accept it or request adjustments.

Method of operation

An estimate may already be set out in the master test plan for the test level. Nevertheless, this remains a necessary step. The test manager has to determine, on the basis of the strategy created, how many hours and possibly how much money will be required. If these exceed the margins of the allocation contained in the master test plan, the test manager should work with the test manager of the overall test process to resolve these discrepancies. Either the strategy or the estimate will need to be amended.

In practice, the number of test hours required almost always is a factor reflected in the estimate. Another, less apparent, part of the estimate is the financial part. How much do those hours cost? Do they involve internal or external resources or even outsourcing? What are the fees? But also: how much do the test environment, test tools and work stations cost? If the client requires it, the test manager also must create a financial budget.

Subsequently, within a test level, the time required for the various phases, such as Planning, Control, Setting up and maintaining infrastructure, Preparation, Specification, Execution and Completion is established. At the start of each test phase, the test manager estimates the effort for the separate test activities.

In more detail

Estimation techniques

The various estimation techniques and the steps involved in arriving at an estimate are described in section 4.10 "Estimation Techniques". For a test plan, estimates can be made based on:

- Ratio figures
- Test object size
- Work Breakdown Structure (WBS)
- Proportionate estimation
- Test point analysis (TPA).

In order to increase the reliability of the estimate, you would be well advised to use your own figures based on experience as well as other means and techniques of estimating.

Tips

- Sometimes a total budget is imposed by the client for testing. This has to be spread across the test phases and the characteristic/object part combinations. Some of the techniques described in section 4.10 (Ratios and Work Breakdown Structure, in particular) provide assistance here. It should also be examined whether the

budget is adequate. The following tips are useful, but much depends on the experience of the test manager:
1) It is best to create an estimate by summing up the characteristic/test intensity estimates (e.g. in PAT, a light performance test + a thorough security test = 120 + 200 hours = 320 hours).
2) Another option is to evaluate the total budget using a rule-of-thumb summing up for test levels: 15% of the total project budget of 5,000 hours for the ST, 20% for the AT = 750 and 1,000 hours respectively.
3) A third option is to employ a standard allocation formula for characteristics, established on the basis of a number of experiences. An example is to give Functionality, at risk category B, 70% of the total budget, at A, 80% and at C, 60%. You can also do this with a fixed number of hours, e.g. User-friendliness could be given 70 hours at category C, and up to 130 at category A (for an average system). This will make it easier to assess the real value of the individual estimates per test level /characteristic.
4) Compare the allocated budget with the budgets and total hours spent in the course of comparable exercises in the past, both in the organization itself and if possible in other, comparable, organizations.

The estimate should be assessed for real value and should come out at around the level of the total assigned budget. Otherwise, adjustments will be necessary, by opting for a higher total budget, or testing fewer characteristics and/or testing with less test intensity.

The figures used above are realistic examples. More information can be found in section 4.10 "Estimation Techniques".

- A test intensity of ••• for a characteristic in a particular test level (e.g. Functionality in the ST), means that the system should be tested more intense than average, not that the entire system should be tested in the greatest possible test intensity. See also the comment under Test Strategy.
- The creation of an estimate for the test has a wide margin of uncertainty. It is important that the test manager make it clear to the stakeholders that the estimate is based on a number of assumptions and may therefore have to be revised later. A possible solution is the use of uncertainty margins. At the beginning of the test, the margin would be, for example, around 40%; at the start of the test execution this becomes around 25%, and somewhere in the middle it becomes around 10%.
- The link between estimate and test intensity (using specific test techniques) is opaque. How much extra time does, for example, the application of the elementary comparison test require as against the data combination test? Few past figures are available for this, and much is done on the basis of the test manager's experience and intuition.
- Various other factors (the quality of the testers, of the test object and test basis, test environment and test tools) can also exert significant influence on the estimate. These factors are either not known at the time the estimate is made or their effect on the estimate is very unclear. The test manager has to make assumptions here, and if necessary include them as assumptions in the plan, and most certainly should evaluate the assumptions as soon as possible.
- It is difficult to 'sell' the required maintenance test effort to management. A general 'testing image' problem is that testing costs too much in management's view. With testing during maintenance, that is reinforced by the fact that testing

has a relatively big share in the maintenance effort – up to as much as 80%. This is partly because the total test costs consist of fixed and variable costs. Fixed costs refer to, for example, the effort required to prepare the test environment, or the execution of a 'standard' regression test; variable costs refer to, for example, the preparation and testing of implemented changes. With the testing of a small change, the percentage of fixed costs is high, since, irrespective of the size of the change, the environment must always be prepared and the regression test run. The greater the changes become, the more the fixed costs decrease in relative terms. For example, with the testing of a change, a 4-hour regression test is always run. If the testing of a change takes 8 hours in total, the fixed testing time amounts to 50% (4/8). If the testing of one or more changes takes a total of 40 hours, then this decreases to 10% (4/40).
In general, the share of testing (fixed+variable) lies between 35% - 80% of all the maintenance activities. It is up to the test manager to make this clear to the client and to put the case for the importance (to the testing) of bigger, controlled releases, in which many changes are bundled, over the implementation of a constant procession of small changes.

Products

The estimate for the test level, in hours and optionally in money, supplied with assumptions used in this, established in the test plan.

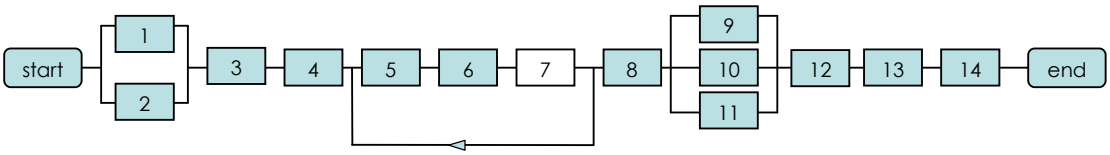
Techniques

Various estimation techniques (section 4.10)
Step-by-step plan for creating an estimate (section 4.10).

Tools

Planning and progress monitoring tools (spreadsheets for estimating the effort and for Test Point Analysis are available at www.tmap.net).

4.3.2.7 Determining the planning



Aim

The creation of as reliable as possible a planning for the test level, so that the client can make allowance for this and can manage accordingly. The principle of the planning is to find the most significant defects (the finding of which belongs within the scope of the test level) first.

Method of operation

Based on the planning of the system development process and on the master test plan, a planning for the test level is created. The test manager indicates the start and end date per phase and the products to be delivered. The planning should cover at least:

- Activities to be carried out (at activity level per phase)
- Correlations with and dependencies on other activities (within or beyond the test level and between the various phases and other test levels)
- Time to be spent per phase
- Required and available resources (people and infrastructure)
- Required and available turnaround time
- Products to be delivered.

Depending on the client's requirements, the financial consequences of the choices made should be made visible in a financial planning. This means, for example, the setting out of the costs in terms of time for the (internal and external) personnel, training, workstations, test environment and test tools.

In more detail

In creating a planning, the following principles apply:

- The test strategy and estimated effort form the basis of the setup of the planning
- In a good planning, the characteristics/object parts designated as high risk are tested as early as possible
- With optimal planning, as far as possible only the test execution activities are carried out on the critical path of the project
- If there are no past figures available in respect of the number of retests, it is advisable to make allowance when creating a planning with one retest on average. Based on past experience, it can be decided to allow more or less time for retesting
- With maintenance, in particular, and with iterative or agile development phases, it is important to make allowance for the execution of regression tests
- When creating a planning, make allowance for the required time of third parties. For example, repair time for defects or time for preparing the test environment
- The transfer of the test object to, and installation in, the test environment often falls between two stools in planning, or rather between the planning of the development and testing activities. Particularly the first few times, this activity

appears to cost significant amounts of time – days rather than hours. Make allowance for this
- Try to streamline the in- and outflow of personnel, so that peaks and troughs in staff levels are avoided.
- Further planning indications can be found in the IT classic [Brooks, 1975/1995].

In more detail

Required information
In order to set up a planning based on the estimated effort, additional information is required concerning the following subjects:

- Available resources
  Worth noting here is that with the estimation of the effort, only limited allowance is made for the available resources. The calculation is made on the number of hours required. In combination with a deadline, this means that a certain number of resources is required for carrying out the planned tests. In practice, it is often the case that the number of available resources initially does not correspond with the required amount of resources. The test manager should make this clear and then discuss it with the client. Possible solutions are the hiring of temporary personnel, extending the timeline or adjusting the strategy.
- Available timeline
  In practice, the available timeline is usually provided in the form of a deadline for the relevant phase.
- Availability of resources, such as test environments and test tools
  When are these to be available for the activities? Do the test tools, for example, still have to be selected, purchased and set up?
- Dependencies between the various activities
  Activities that depend on other activities can only start after completion of those other activities and not in parallel with them.
- Method of system development
  The test levels are planned depending on the way in which the system is developed. With a waterfall method, the phasing is different from that of an iterative process in which testing and development activities are parallel and sometimes executed integrally. The development test and system test as a rule have more to do with this than the acceptance test.
- Information on milestones in the development project
  This information is necessary in order to coordinate the test planning optimally with the planning of the rest of the project. This makes it possible to minimize the total timeline of the project.

The planning is reflected in, for example, a network planning or a bar chart, depending on the method used within the organization. This book does not deal with planning techniques, because for the test process the test manager employs standard planning techniques that are not specific to testing.

Example of activities planning

TEST PHASE	**Week number (2006)**														**Hours/FTE**
	14	**…**	**20**	**21**	**…**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	
Planning, Control and	x	x	x	x	x	x	x	x	x	x	x	x	x	x	2 FTE
Setting up &															

maintaining infrastructure															
Preparation and Specification	x	x	x	x	x	x	x								3480 hours
Exec. FAT, FIT (functional integration test)								x	x	x	x				3480 hours
Exec. CT (chain test)										x	x	x	x		
Exec. UAT										x	x	x	x		
Completion														x	
Spare week														x	
													Total:		**2 FTE + 6960 hours**

Example of milestone planning

Milestone	**Date**	**Owner**
Delivery of definitive test basis	01-03-2006	SAP project leader
Delivery of test infrastructure	31-08-2006	SAP project leader
Delivery of test object	31-08-2006	SAP project leader
Completion of FAT, FIT, CIT, UAT test specifications	31-08-2006	Test coordinator
Completion of test execution	14-10-2006	Test coordinator
Delivery of testware	22-10-2006	Test coordinator
Delivery of Preliminary Release advice	15-10-2006	Test manager
Delivery of Release advice	22-10-2006	Test manager
Delivery of Test Report	22-10-2006	Test manager

Tip

When planning resources, indicate from which point it is no longer possible to accommodate imminent overrun by deploying extra people. Sometimes an environment is so complex or specific that an 'extra hand' will no longer gain time. It is not pleasant to have to explain this when the moment has already arrived and the project leader is already busily engaged in arranging extra people for the test team – even less so when those extra people are already being introduced to the team…

An aspect of planning related to quality is when a test level is ready and the test object can be transferred to the following test level or to production. In other words, what can the 'next' test level expect after the 'previous' test level is completed. In order to make these expectations explicit, requirements are set according to the result of the test level. In practice, these requirements are also known as exit criteria. With increasing outsourcing, it becomes more and more important to establish clear exit criteria to prevent the supplier from delivering inadequate quality.

In more detail

Exit criteria can relate, for example, to the number of issues in a particular risk category that may still be open, the way in which a certain risk is covered (e.g. all the system parts with the highest risk category have been tested using a formal test design technique), or the intensity with in which the requirements should have been tested. From within the master test plan, the exit criteria are applied to the test level. If that is not the case, or if there is no master test plan, the test manager should agree the criteria with the client.

The box below shows a number of concrete examples of exit criteria:

System X may only be transferred to the AT when the following conditions have been met:

- There are no more open defects in the category of "severe"
- There is a maximum of 4 open defects in the category "disrupting"
- The total number of open defects is no more than 20
- A workaround has been described for every open defect
- For every user functionality, at minimum, the correct paths have been tested and approved

System X may be transferred to the AT when it can be shown in writing that all the risks that were allocated to the ST in accordance with document Y have been tested in the agreed test intensity and by the agreed test method.

An important point of focus as regards the above-mentioned criteria is that clear definitions should be agreed by all the stakeholders of what a particular category of severity is and what is meant by 'agreed test intensity and test method'. In practice, a lack of clarity here can lead to heated discussions.

In more detail

Similarities and differences between acceptance and exit criteria
Another term for exit criteria that is used is 'acceptance criteria', as discussed in subsection 6.2.2. Besides the fact that acceptance criteria may be a broader term than exit criteria, another difference is that acceptance criteria come at the end, i.e. at acceptance, and exit criteria at the transfer from one test level to another, or to production. Figure 43 illustrates this.

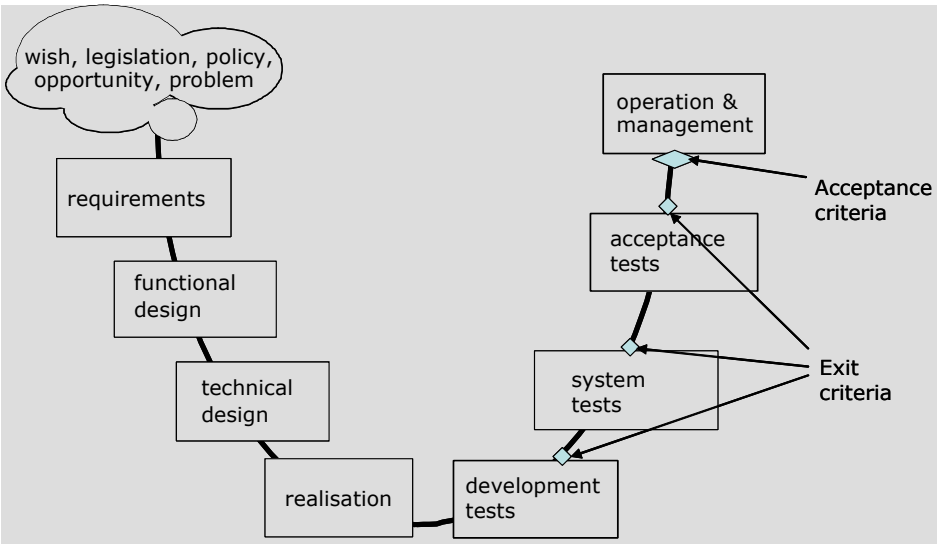


Figure 43. Exit and acceptance criteria

Tip

Suspend and resume criteria
In some, particularly formally set up, tests, so-called suspend- and resume criteria may be defined in the plan. These criteria indicate under which circumstances the testing is temporarily suspended and then resumed. Examples of suspend criteria are that testing has to stop when a particular infrastructural component is not available, or if a test-blocking defect is found. A resume criterion may be that with the lifting of the suspend criterion the testing of the system part /function/component has to take place entirely anew.

Feedback
When the test manager has created a planning, this is the time to agree matters with the client. If the test strategy setup and subsequent estimate of required effort and planning are not acceptable, then these steps are repeated. With this, the client and test manager consider whether to test certain aspects with lesser test intensity, so that time and/or money is spared, but a higher level of risk is accepted, or the other way around. To facilitate communication, the test manager refers here to the original test goals. Where a master test plan exists, the coordinating test manager is involved here, but the client makes the final choice.

An adjusted strategy is illustrated below, with less test intensity indicated by ○ instead of ● and more test intensity by ●.

ST example

Characteristic	RC MTP	ST MTP	Subsys1	Subsys2	Total sys
Functionality	n/a	n/a	A/●●○ functional, regression	B/○● functional, regression	C/● integration, multi-user
Performance online	B	●	-	-	C/●

					random testing in ST environment
…					

UAT example

Characteristic	**RC MTP**	**UAT MTP**	**Subsys1**	**Subsys2**	**Total sys**
Functionality	n/a	n/a	A/●○ functional	B/-	C/● regression
User-friendliness	B	●●	C/I		B/●○ usability
Security	A	●			
- authorization matrix	B		-/E authorization test	-/E authorization test	B/●● process test
- application	C		-	-	C/● penetration test
Suitability	B	●●●	B/●● scenario test	C/● scenario test	A/●●○ process test
…					

The adjusted strategy leads to another estimated effort and planning, and also to an indication of bigger (or even smaller) product risks, translated into terms that are comprehensible to the client (referring back to the product risk analysis with test goals, characteristics and object parts).

In addition to the feedback on strategy, budget and planning, the test manager discusses with the client the use of tolerances in the execution of the test process. These are boundaries within which the test manager is not required to ask the client's permission. For example, a tolerance of 5% is often agreed for the budget. For the planning, it may be agreed that only deviations from project milestones will require discussion. With strategy tolerances, for example, the client's advance permission is not required for testing a characteristic/object part with a greater or lesser test intensity.

Products

Planning for the test process
Exit criteria
Optional: tolerances for strategy, budget and planning
Optional: suspend and resume criteria
(above products are established in the test plan)
Strategy, budget and planning feedback to/from the client.

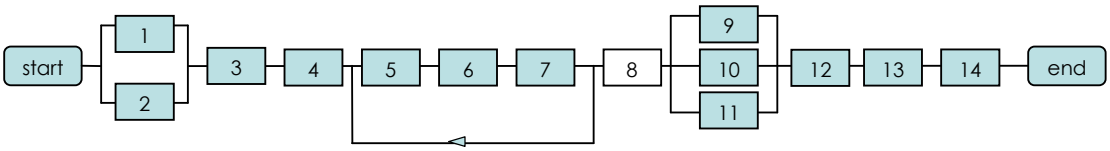
Techniques

Not applicable.

Tools

Workflow tool.

4.3.2.8 Allocating test units and test techniques



Aim

To finalize the test types and the thorough/light testing of characteristics/object parts based on the approved test strategy, budget and planning.

Method of operation

The method covers the following subactivities:
1. Determining test units
2. Allocating test techniques

This step requires information that is not always readily available in practice. In that case, the test manager will carry out this step in a general manner and bring in the details at a later stage, during the phase "Control".

1) Determining the test units

Within the strategy, test types and test intensity are allocated to the characteristics/object parts. In some cases, a test type may be very extensive for a particular characteristic/object part. To facilitate the definition of manageable and executable activities, the test manager splits the object part further into 'test units'.

Definition A test unit is a collection of processes, transactions and/or functions that are tested collectively.

The advantage of a test unit is that it forms a manageable unit (X hours in Y period) and as such, it is an important management mechanism for the test manager. Reasons for splitting a object part into test units are:

- The size of the object part is too big to be able to manage the testing of it effectively
- A particular piece of the object part requires a separate test method of operation with other test techniques, e.g. because the risk strongly deviates or because the nature of the part deviates from the rest (screen as against processing).

Since a test unit represents a unit of work, it is advisable for the test manager to coordinate this with the developer, so that a delivery unit corresponds with one or more test units and no half-test units are delivered.

2) Allocating test techniques

A subsequent step is that, per test type and based on the chosen test intensity, one or more suitable test techniques are selected with which the test is to be specified and executed. If a object part is divided into test units, techniques are allocated per test unit. But how, then, do you select the suitable techniques? Chapter 3 "Website" covers approaches, coverage types and test design techniques. Variations can be made on

these, and there are other techniques, including those you create yourself. Checklists, too, can be used as a technique. This choice is, besides the chosen test intensity, strongly dependent on a number of other aspects:

- Test basis
  Are the tests to be based on requirements; is the functional design written in pseudo-code or easily converted to it; are there state-transition diagrams or decision tables, or is it very informal with a lot of knowledge residing in the heads of the domain experts? Some techniques rely heavily on the availability of a certain form of described test basis, while with others the test basis may be an unstructured and poorly documented collection of information sources.
- Test type / quality characteristics
  What is to be tested? Some test design techniques are mainly suitable for testing the interaction (screens, reports, online) between system and user; others are more suitable for testing the relationship between the administrative organization and the system, for testing performance or security, or for testing complex processing (calculations), and yet others are intended for testing the integration between functions and/or data. Checklists are also often used for testing non-functional quality characteristics. All of these relate to the type of defects that can be found with the aid of the technique, e.g. incorrect input checks, incorrect processing or integration errors.
- What kind of variations should be covered, and to what degree?
  Which test intensity is required? This should be expressed by defining one or more coverage types.
- Knowledge and expertise of the available testers
  Have the testers already been trained in the technique, are they experienced in it or does the choice of a particular technique mean that the testers need to be trained and coached in it? Is the technique really suitable for the available testers? Users are normally not professional testers.
- Labor-intensiveness
  How labor-intensive are the selected techniques, and is this in proportion to the estimated amount of time? Sometimes other techniques should be chosen, with possibly different coverage, to remain within budget. If this means less thorough testing is to be carried out than was agreed, the client should, of course, be informed!

After having covered the above aspects, the test manager makes a selection of techniques to be used. An example is set out below:

Example

Characteristic	**Object part**	**Test type**	**Techniques**
Functionality	Subsys1 (A/●●)	Functional test	tu1: DCoT tu2: SYN, SEM
Functionality	Subsys1 (A/●●)	Regression test	tu3: selection from tu1 and tu2
Functionality	Subsys2 (B/●)	Functional test	tu4: Exploratory Testing tu5: SYN, SEM
Functionality	Subsys2 (B/●)	Regression test	tu6: selection from tu4 and tu5
Functionality	Total system (C/●)	Integration	tu7: DCyT
Functionality	Total system (C/●)	Multi-user	tu8: Exploratory Testing

Performance online	Total system (C/●)	Random test in ST environment	tu9: Error Guessing
…			

In this example, the testing of subsystem 1 is spread across test units ("tu") 1 and 2; subsystem 2 consists of test units 4 and 5. Test unit 1, with many complex calculations, is nevertheless given rather an light technique, with the Data Combination Test (because the client opted for an average test intensity), test unit 4 contains processing functionality and is allocated the (very free) "technique" of Exploratory Testing; test units 2 and 5 consist mainly of screens and are each given 2 techniques: the Syntactic and Semantic Test. The total system is then tested for coherence with the Data Cycle Test (test unit 7) and the multi-user aspect with Exploratory Testing (test unit 8). Later regression tests consist of a selection of previously created test cases (test units 3 and 6). Finally, a light Performance test is carried out using Error Guessing (test unit 9).

UAT example

Characteristic	**Object part**	**Test type**	**Techniques**
Functionality	Subsys1 (A/●)	Functional test	tu1: ST random test tu2: ST random test
Functionality	Total system (C/●)	Regression	tu3: DCoT
User-friendliness	Subsys1 (C/I)	User-friendliness	implicit in tu1 and tu2
User-friendliness	Total system (B/●)	Usability	tu4: SUMI
Security – auth.matrix	Subsys1 (-/E)	Authorization test	tu5: auth. matrix random test
Security – auth.matrix	Subsys2 (-/E)	Authorization test	tu5: auth. matrix random test
Security – auth.matrix	Total system (B/●●)	Process test	tu7: SEM
Security – application	Total system (C/●)	Penetration test	tu8: Error Guessing
Suitability	Subsys1 (B/●●)	Scenario test	tu9: PCT, test depth level 2
Suitability	Subsys2 (C/●)	Scenario test	tu10: PCT, test depth level 1
Suitability	Total system (A/●●)	Process test	tu11: PCT, test depth level 2

In this example, use is made in test units 1 and 2 of ST test cases. The regression test on the total system takes place with the light Data Combination Test. User-friendliness is implicitly tested simultaneously with test units 1 and 2 by evaluating the testers' impressions after completion. Thereafter, an explicit test takes place with the aid of the SUMI checklist. The authorization matrix is first randomly checked for correct input, and then the authorizations are tested explicitly using the Semantic Test. A light penetration test takes place using Error Guessing, and Suitability is tested using the Process Cycle Test.

If the decision has been made to perform explicit testing, the table below can provide assistance in selecting the test design techniques to be employed. Per quality characteristic, the table provides various test design techniques that are suitable for testing the relevant characteristic. This table can also be found at www.tmap.net.

For the relevant quality characteristics, usable test design techniques are mentioned, making a distinction with respect to the thoroughness of the test. • means light, •• average and ••• thorough. The techniques mentioned should be seen as obvious choices and are intended to provide inspiration. The table is certainly not meant to be prescriptive – other choices of techniques are of course allowed.

Quality characteristic	**Test design technique**		
	• / light	**•• / average**	**••• / thorough**
Manageability - installability	CKL	DCoT	DCoT
Security	CKL	SEM	Penetration test
Usability	UCT	UCT PCT*	RLT
Continuity		RLT	RLT
Functionality - integration	DCoT	DCoT GCT PCT*	DCoT
Functionality - detail	DCoT	DCoT EVT	DCoT + boundary values EVT + boundary values DTT
Functionality - validations	SYN	SYN SEM	
User-friendliness	SYN	SYN UCT* PCT*	Usability test (if necessary in lab)
Infrastructure (suitability for)		RLT*	
Suitability	PCT test depth level-1 UCT*	PCT	PCT test depth level-3
Performance		RLT	
Portability	CKL Random sample functional tests Random sample environment combinations	Functional regression test Important environment combinations	All functional tests All environment combinations
Efficiency		RLT	

Notes on the above table:

- Abbreviations used:

* If the technique is adapted to some extent, this can be used to test the relevant quality characteristic
DTT Decision table test
CKL Checklist
DCoT Data combination test
DCyT Data cycle test
ECT Elementary comparison test
DCT Data cycle test

PCT Process cycle test (depth level = 2)
RLT Real-life test
SEM Semantic test
SYN Syntactic test
UCT Use case test
For a comprehensive description of these techniques, please refer to section 3.7 "A basic set of test design techniques".

- Concepts used

Environment combinations
In portability testing, it is examined whether the system will run in various environments. Environments can be made up of various things, such as hardware platform, database system, network, browser and operating system. If the system is required to run on 3 (versions of) operating systems, under 4 browsers (or browser versions), this runs to 3 x 4 = 12 *environment combinations* to be tested.
Penetration test
The penetration test is aimed at finding holes in the security of the system. This test is usually carried out by an 'ethical hacker'.
Portability – functional tests
In order to test portability, testing random samples of the functional tests – in increasing test intensity – can be carried out in a particular environment, the regression test or all the test cases.
Usability test
A test in which the users can simulate business processes and try out the system. By observing the users during the test, conclusions can be drawn concerning the quality of the test object. A specially arranged and controlled environment that includes video cameras and a room with two-way mirror for the observers is known as a usability lab.

In more detail

Test design techniques are actually first linked to test types and then to quality characteristics. In the absence of an unambiguous set of test types, a direct link to quality characteristics is selected.

The techniques of Exploratory Testing and Error Guessing do not appear in the above table. The reason for this is that these techniques can be used for all quality characteristics. Exploratory Testing is any form of testing with the tester making his test design during the execution of the test. The information obtained in the course of testing is used to design new and improved test cases. Error Guessing means that testers test the system in an unstructured way.
Since it is impossible to establish all possible test situations in test cases, Exploratory Testing and Error Guessing are valuable techniques for carrying out supplementary testing. It is advisable to allow a limited amount of time during each test period for these techniques.

If a cell is left empty and no obvious techniques have been cited, then Error Guessing or Exploratory Testing can be applied. If relevant techniques have been cited, such as with Functionality validations with in-depth coverage, then aforesaid techniques can be used as a basis. These techniques can often be executed with a deeper-level variant, or more than just one technique can be selected.

In more detail

Much uncertainty
In some cases, there is much uncertainty as to where the risks lie. This makes it difficult to determine a good strategy and to choose the right techniques. There are two possible solutions to this:

- Exploratory testing, because this has the flexibility to zoom in as necessary during the test execution on where the risk areas appear to be
- Employing the "onion" model. With this, somewhat general tests are specified in advance, but time and budget are planned for creating additional and targeted deeper-level tests during the test execution as the areas of risk become clearer. The testing thus progresses to a deeper layer each time.

Products

Overview of test units with allocation of test techniques.

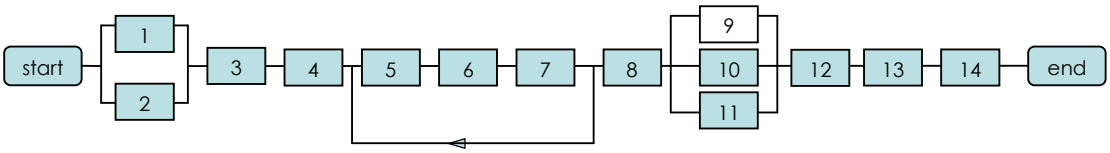
Techniques

Not applicable.

Tools

Workflow tool.

4.3.2.9 Defining the test products



Aim

The clear definition of the test products to be delivered.

Method of operation

The activities that are carried out for the purposes of planning, execution and managing the test process deliver certain products, such as the test plan and reports, test cases and test scripts, but also procedures, instructions and project documentation, such as consultation notes. In consultation with the client and other stakeholders, it is determined what products are to be delivered. If there is a master test plan, this will also define test products to be delivered. This may concern testware, such as test plans, test scripts or automated regression tests – products, therefore, that are eligible for reuse, but which also may define test documentation, such as progress reports.

Tip

The use of tools for configuration or test management helps to produce a uniform method.

The following test products can be distinguished:

- Testware

Definition

Testware is all the test documentation produced in the course of the test process that can be used for maintenance purposes and that should therefore be transferable and maintainable.

In retests or regression testing, existing testware is often used. Testware covers, for example:

- Test plan(s)
  Includes both master test plans and other test plans.
- Logical test specifications
  The logical specifications contain the logical description of the test cases.
- Physical test specifications
  The physical test specifications contain the physical description of the test cases and the test scripts. Physical means that the test cases are executable and checkable. The physical test cases are converted from the logical test cases. A test script contains the physical test cases placed in the most efficient order of execution.
- Traceability matrix (or cross-reference matrix)
  A matrix in which the link is indicated between the test basis (requirements, functional specifications, etc.) and the actual test cases. The situations to be tested from the test basis are shown vertically and the test cases horizontally.
- Test input files
  The test input files created on the basis of the test scripts should contain a

(brief) description of the following:
    - Aim
    - The "physical" name
    - Date created
    - Brief description of the content
    - The file type and other relevant features
    - Reference to the test scripts.
  - Basic documentation
    A description of the test environment, test tools, test organization and underlying databases.
  - Test execution dossier
    The test execution dossier consists of:
    - Test results (logging of executed tests and test cases) and reports
    - Test execution (optional)
      The "material evidence" of the executed tests can consist of screen dumps, print output and output files. After completion of the test, the tester delivers the produced output to the administrator. The test documentation to be delivered from the output contains:
        - Reference to the "physical" name
        - Date of creation
        - Brief description of the content
        - File type and other relevant features
        - Reference to the test script.
    - Information on the defects and the changes
    - Transfer and version documentation.
- Other test (project) documentation
  During the test process, various documents are received or created that are not meant for reuse, such as:
  - Project plans
  - Reports of the discussions (with lists of decisions and activities)
  - Correspondence, both on paper and electronic (e-mail)
  - Memos
  - (Project) standards and guidelines
  - Test, review and audit reports
  - Reports on progress and quality
  - Etc.

By means of a brief description, the content and the aim of the various products or documents are indicated. Besides listing the products to be delivered, norms and standards can also be supplied and reference made to templates.

Products

A description of the test products to be delivered including norms and standards and any reference to templates, established in the test plan.

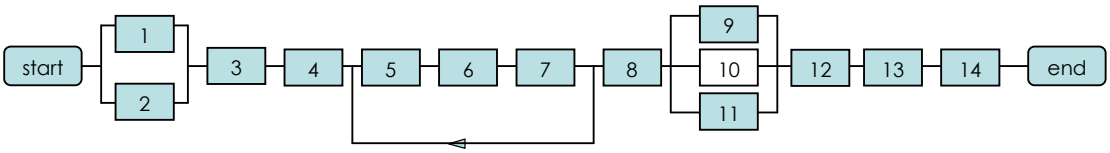
Techniques

Not applicable.

Tools

Testware management tool.

4.3.2.10 Defining the organization



Aim

To define the roles, tasks, authorizations and responsibilities that are applicable to the test level.

Method of operation

The method covers the following subactivities:

1. Determining necessary roles
2. Delegating tasks, authorizations and responsibilities
3. Establishing the organization
4. Allocating personnel
5. Establishing training and coaching needs
6. Establishing communication structures and reporting lines.

1) Determining necessary roles

In order to facilitate the activities in the test process, the test manager determines which test roles are required and how they are to be filled. These will include, for example:

- Test management or test coordination
- Test team leading
- Tester
- Management (test process, test products, defects)
- Intermediary
- Support (domain knowledge, system knowledge, test environment, test tools or test method).

Make as much use here as possible of the roles set out at coordinating level.

2) Delegating tasks, authorizations and responsibilities

The tasks and responsibilities are set out here per required role.

Examples of tasks (with the most likely role shown in parenthesis):

- Creating and maintaining the test plan (test manager)
- Directing the execution of, monitoring and adjusting the test activities (test manager)
- Carrying out a testability review on the test basis (tester)
- Designing tests based on user information (tester)
- Specifying test cases and test scripts (tester)
- Executing tests (tester)
- Organizing automated test execution (test-tool specialist, test-tool programmer)
- Organizing the technical infrastructure and the management of this (test infrastructure coordinator)
- Organizing methodical, technical and functional support (test manager)
- Reporting on the test progress and quality of the test object (test manager)

- Supporting users in creating test cases (specifically for iterative development) (tester)

3) Establishing the organization

The correlations between the roles mentioned and the relationships with the other stakeholders within the system development process have to be determined and established. The organization of the test level naturally forms part of a bigger whole. If the whole is a project, the test manager should also establish a relationship with the test or quality department, if any.

For the organization of a test level, the possibilities can largely be defined as follows (see figure 44):

1. Testing as an **independent activity** or **integrated with other activities**
2. Testing placed within a **project** or in a **line organization**

These choices are dependent on the test level, project and organization. Sometimes, but by no means always, the test manager can exert influence on this.

	independent activity	**integrated**
project organisation	acceptance test, traditional system test	development tests, system test in agile environment
line organisation	testing factory	maintenance process

Figure 44. Organizational divisions with examples

Below are the most significant organizational forms with a few examples briefly mentioned. The descriptions and advantages are emphatically meant as a general indication; there are often exceptions in practice.

In more detail

Testing as an independent activity in a project

Within the project, a team is responsible for organizing and executing the test. The testers within the team as a rule have a lot of test knowledge, together with – depending on the test level – a mix of system and organizational knowledge.

Advantages:

- Good accessibility to knowledge of the system
- Good coordination among users, developers and testers
- Knowledge and skills of testers are easily discernible
- Focus on an aim, therefore more manageable
- Independent assessment of the quality of the test object.

Testing integrated within a project

Within the project, testers, users and developers work in the same team. There are often several teams in action. The tester is responsible within his team for the organization and

execution of the test. The tester, as a rule, has a lot of technical knowledge of the system and architecture.
Advantages:

- Excellent knowledge of the application and architecture
- Close co-operation among users, developers and testers
- Very short lines of communication
- Focus on an aim, therefore more manageable

Testing as an independent line organization
A separate department or organization has testing – both the organization and execution – as its primary task. Projects or other line departments issue a certain test instruction to this department/organization. Test knowledge is dominant.
Advantages:

- Knowledge and skills of testers are easily discernible
- Independent assessment of the quality of the test object
- Efficiency gain through reuse and test automation
- Permanently set up infrastructure facilitates a fast start
- Standard test process setup facilitates a fast start
- Increased motivation through career prospects of testers

Testing integrated in the line organization
Within a development or system management department, the role of tester is often combined with other roles. The tester in this organizational form often has a lot of system and/or organizational knowledge.
Advantages:

- Excellent knowledge of the system and the organization
- Close co-operation among users, developers and testers
- Short lines of communication
- Knowledge management concerning an application is easier to realize

Below, some examples of organizational forms are set out.

Example

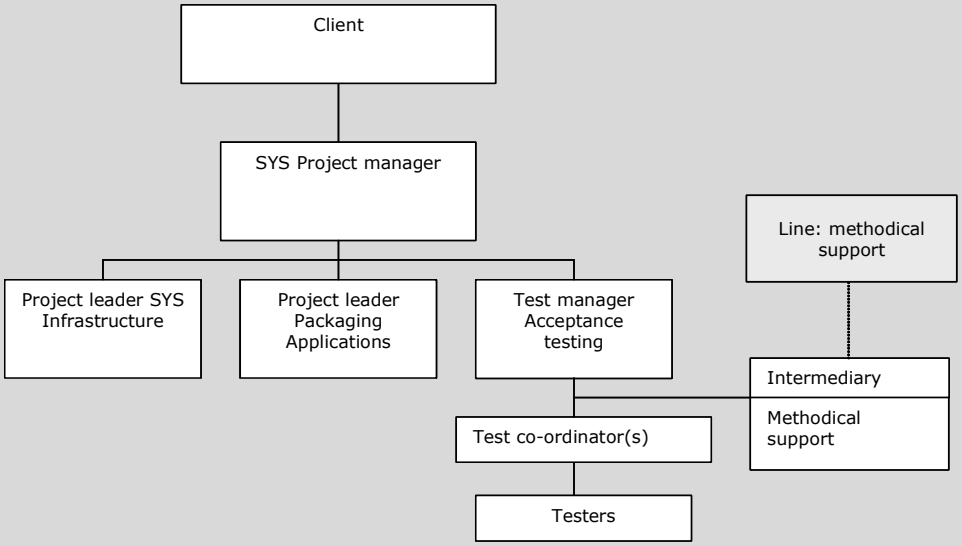


Figure 45: Example traditional organization

The above is a rather traditional organization where the acceptance test manager falls under the project manager and the system test under the project leader (Packaging Applications). Test support is supplied from within the line.

Example

In the example below, the test manager comes under both the project manager of the SAP system and the project manager who has to implement it in the department. While answering to two clients may be an undesirable situation, in this practical example it has gone well. The test manager has stipulated at the beginning that if the two clients disagree, they should resolve their differences without involving the test manager. There has been no incidence of this.

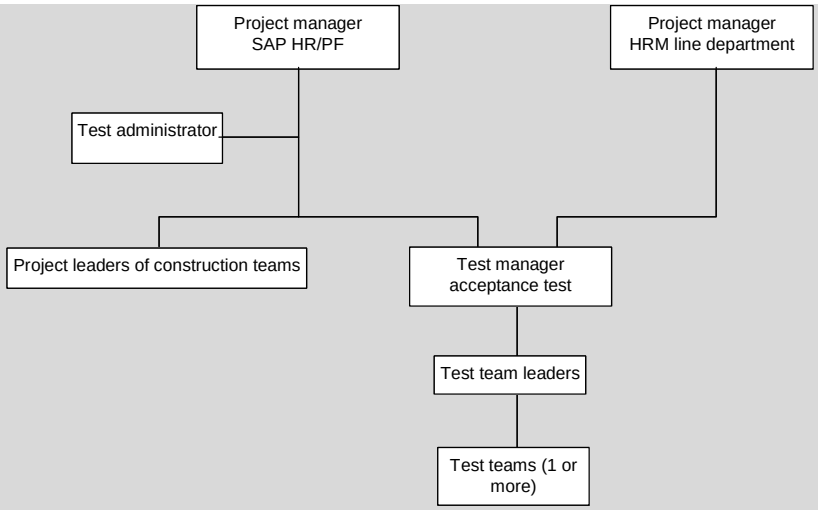


Figure 46: Example organization with two managers

Tip

Iterative and agile system development
A disadvantage of integrated testing cited is that it can impair the independent quality assessment of the tester. A possible solution to this is to place the test manager apart from the development teams, with the testers in these teams answering to him (see figure 47). The advantage is that the gaps in the teams between developer, user and tester remain as small as possible, while the test manager can be alerted if the testing within a team runs into difficulties through planning pressure or other circumstances. This requires insight into the total product and project on the part of the test manager, combined with a good political feel for the balance between 'quality' and 'meeting deadlines'.

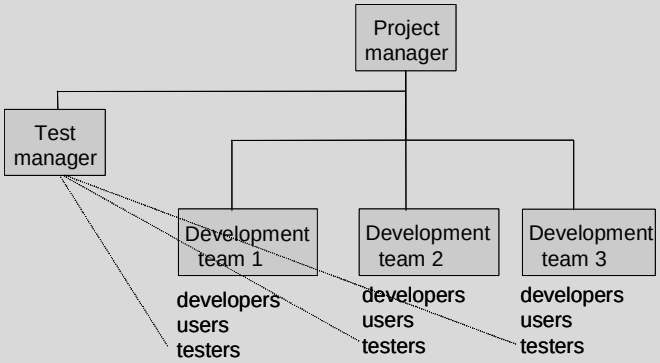


Figure 47. Example test management apart from teams

In more detail

RACI
If necessary, a RACI table could be set up, showing activities and stakeholders set out against each other. RACI stands for Responsible, Accountable, Consulted and Informed.

At every crossroads, it can be indicated whether a party is directly responsible (R), is accountable (A), should be consulted (C), informed (I), or not at all.

It is impossible to determine one preferred organization for testing. In general, the structure of the test organization should resemble that of the associated process of system development or package implementation. In many cases, this means the project organization. If there is to be frequent (re)testing in combination with scarce (test) knowledge, the permanent test organization discussed in section 8.3 becomes a candidate.

4) Allocating personnel

When it has been established which test roles should be filled within the test process, the test manager delegates people to the roles. In this, of course, he makes allowance for their availability and skills in relation to the knowledge and skills required in the relevant roles (see section 8.6 "Test professional" and Chapter 16 "Test roles"). For the sake of clarity: the roles do not have to be filled by test professionals; end users or developers, for example, may be assigned the role of tester. The important thing is that the team as a whole has the right mix of knowledge and skills in the area of system, organization and testing. By the way, one individual can take several roles, but it must then be ensured that this does not result in conflicting responsibilities.

Tips

- To have more certainty that the testers have sufficient test knowledge, one can ask for certified testers. EXIN (Examination Institute for Information Science) organizes a certification scheme specifically for TMap. The ISTQB (International Software Testing Qualifications Board) is responsible for an international qualification scheme for testers.
- When people are deployed from other departments, or even other organizations, the test manager should make allowance for agreements, procedures, selection processes, etc. This can take up a lot of time.
- There is often external pressure to accept certain people as testers into the team. If these people are not suitable, the test manager should be firm and spell out the consequences in terms of high training and coaching costs and low productivity.
- Employing or hiring in a tester cannot simply be left to a personnel or purchasing department. Good information on this can be found in [Rothman, 2006].

Besides the suitability of the individual for the role(s), there is a further dimension: that of the team. The natural inclination of the test manager is to select those persons whose personality most appeals to him. This can result in a team of similar characters. The theory of team formation teaches that, in fact, it is the team with a mix of personalities that achieves the best results. Possibly the best-known model in this area is the 9 team roles of Belbin (see also www.belbin.com). This differentiates between functional, organizational and personal roles. The ideal composition of each team depends on the aims.
Belbin distinguishes the following roles, with a number of characteristics per role:

Plant	Creative, individualistic, imaginative, intellectual, knowledgeable
Chairperson	Calm, self-confident, sober, purposeful, brings out the best in every team member
Monitor/evaluator	Has strategic insight, is sober, unemotional, analytical and critical
Implementer	Conscientious, conservative, converts decisions into tasks, practical, self-disciplined

Finisher	Painstaking, concerned, works behind the scenes
Resource Investigator	Extrovert, seeks out new possibilities, enthusiastic, communicative
Shaper	Dynamic, energetic, extrovert, impatient
Team worker	Sociable, co-operative, listens well, encourages and integrates
Specialist	Professional, solo player, dedicated

For broader theory on this, refer to [Belbin, 2003]. A translation into the best test-team composition is provided [Lloyd Roden, 2005].

5) Establishing training and coaching needs

The people involved in the test levels should have various types of knowledge, particularly in the areas of testing, domain knowledge and system.

- For testing, this may include: (the advantages of) the test method of operation, strategy determination, test techniques and tools to be applied
- For domain knowledge, bear in mind, for example, the organization and its business processes
- System knowledge may consist of knowledge of the development or implementation process, design techniques, technical architecture, database or programming tools, etc.

In more detail

Knowledge input
The intention is not to include only very experienced testers with extensive knowledge in all 3 areas. Depending on the test level and the composition of the team, each individual will require to have a certain mix of these types of knowledge. If their knowledge is insufficient in one of the areas, it will have to be brought up to the required level. Training is the most obvious answer here, and a budget should be reserved for it. Timing is important: training is most effective if the knowledge gained can be quickly put into practice afterwards. Following any training given, people with insufficient knowledge should be coached in the beginning by someone with experience. This accelerates the learning process considerably. It often takes place "on-the-fly" during the test process, but if it is estimated to be a substantial activity, it should be planned for and hours made available for it.

6) Establishing communication structures and reporting lines

From within the test process, communication takes place with various parties. Examples of the parties with whom the test manager communicates are:

- Client
- Test manager of the overall test process
- Project management (including Change Control Board)
- Acceptors (user organization, system administration, functional management)
- Steering group
- Project leaders (design, construction and/or implementation)
- Developers
- Testing line organization
- Quality management, QA
- Accountancy, EDP auditing.

It should be agreed with each party whether consultation and/or reporting is to take place, and what the aims and frequency of these should be.

Meeting types

For every type of meeting, it should be agreed who will be present and what, if any, the standard agenda will be.

Examples of meeting types for use by the test manager are:

- Weekly meeting with all the other test managers, lead by the test manager of the overall test process
- Weekly project meeting
- Weekly Change Control Board meeting
- Defects meeting (1 x per week as standard; 3 x per week during test execution)
- Weekly test team meeting
- Daily stand-up meeting.

Example of a fixed agenda for a test team meeting:

Agenda item	*Subject*	*Time*	*Who*
1.	**Opening** - **Establish the agenda** - **Announcements**	**xx.xx – xx.xx**	**<test manager>**
2.	**Minutes of meeting dated: <xx-xx-xxxx>**	**xx.xx – xx.xx**	**All**
3.	**Action list dated: <xx-xx-xxxx>**	**xx.xx – xx.xx**	**All**
4.	**Status, progress and quality:** - **test unit 1** - **test unit 2** - **test unit 3** - …	**xx.xx – xx.xx**	**<Tester 1>** **<Tester 2>** **<Tester 1>** **…**
5.	**Quality of test process** - <What is going well and what could be improved?> - <TPI status> - <Defects management> - <Testware management> - <Reviews> - …	**xx.xx – xx.xx**	**All**
6.	**Questions before closure**	**xx.xx – xx.xx**	**All**

Reports

According to the BDTM vision, reporting takes place on the four aspects Result, Risks, Time and Costs.

- Result
    - The outcome of the tests executed at the level of characteristic/object part
    - The result in terms of obtained/not-obtained test goals (business processes, user requirements, etc.)
    - Any trend analyses
- Risk

    - Notification of parts that are being tested more superficially (or not at all) than the risk estimate indicates, thus presenting a higher risk
    - Observed (test) project risks
- Time + Costs
    - Progress of testing (in activities, products, hours spent and, optionally, money, dates)
    - Indication of when the testing will be completed.

Reporting on risks and results takes place at the level of test goals, as agreed with the client and other stakeholders. The risk tables of the product risk analysis are maintained with this aim. It is up to the test manager to translate test results on characteristics/object parts effectively, and on the basis of the tables, to this level.

Reporting can take place in various ways, to various target groups and at various times. The most important forms of reporting are:

- Progress and quality reports
  Information and advice on progress (and, optionally, quality) of the test process and on quality/risks of the test object, based on the four BDTM aspects.
  Frequency: periodically, preferably weekly
- Risk report
  With certain (project) risks, the test manager can, either upon request or at his own initiative, report on risk, the consequences for the test process and possible measures for dealing with the risk. In the Prince2 project management method these are known as 'exception reports'.
  Frequency: ad hoc
- Release advice
  Information and advice on quality/risks of the test object + formally established release advice.
  Frequency: towards the end of the test execution, before the decision has to be taken on release
- Final report
  Evaluation of the test process and test object, looking back from the original plan.
  Frequency: once, at the end of the test process.

The test manager will determine, for each of these forms of report, to whom they should be sent, whether for approval or for information, with what content and degree of detail and with what frequency. In the activity, "Understanding the assignment" the test manager has already looked at which parties should, or wish to, receive reports. In consultation with the client, that is now determined in more detail. As an aid in overseeing who should receive which report, a matrix can be set up of report forms and target groups.

In more detail

In terms of content, the progress and quality report is of the most importance, since it provides information and recommendations, on the basis of which timely management adjustments can be made. The data for this are supplied through management setup. The report should contain details on the most recent reporting period and cumulative data on the entire test process.

Products

A description of the test organization, established in the test plan.

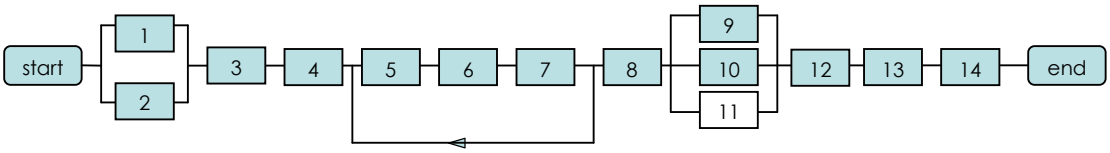
Techniques

Not applicable.

Tools

Not applicable.

4.3.2.11 Defining the infrastructure



Aim

To establish the infrastructure that is required for the test process.

Method of operation

The method of operation covers the following subactivities:

1. Defining the test environment
2. Defining the test tools
3. Defining the office setup
4. Establishing infrastructure planning

1) Defining the test environment

Each test level requires a test environment in order to execute the tests. This environment is generally composed of the following components:

- Hardware
- Software
- Interfaces
- Environment data
- System management tools
- Processes.

The environment should be composed and set up in such a way as to facilitate, on the basis of the test results obtained, the best estimate of the degree to which the test object meets the set requirements. The environment has a considerable influence on the quality, duration and costs of the test process.
In order to manage the test environment effectively, it is often separate from the development or production environment. Moreover, each test level sets its environment different requirements.
At www.tmap.net, a checklist "Test environments" is available that can be of assistance in defining the test environment.
If the test environment already exists, for example in a maintenance process, it may be sufficient to refer to this and to mention any adjustments to be made.

2) Defining the test tools

It is established which test tools are required. Test tools can provide support with most test activities.
Besides the familiar test tools, such as test management, record&playback and defect management tools, you should also think of small, freeware or even self-built tools. Such tools can often be implemented for a small investment in time, but can be extremely valuable. The Internet is invaluable for seeking out freeware tools (search, for example, for "freeware test tool"). For self-built tools, it is advisable to consult the developers; they often already have such tools, otherwise they may be able to make them with very little effort.

In more detail

Since tools are to support the test process, the logical sequence would appear to be to define the process first and then select the tool: "structure before tool". However, this is not entirely true. Some very useful tools (test management and record&playback in particular) set requirements as regards process, e.g. the way in which test cases are established. If the test manager makes no allowance for this, the tool cannot be (efficiently) employed. It is therefore preferable to carry out process setup and tool selection more or less simultaneously.

3) Defining the office setup

The office infrastructure required for testing (workrooms, meeting rooms, telephones, PCs, network connections, office software, printers, etc.) is defined in outline. This concerns an office setup in the widest sense, since testers, too, need to carry out their work in the right circumstances. A checklist for the office setup can be found at www.tmap.net.

The appropriate and timely setting up of the office infrastructure will mean that all kinds of efficiency losses, such as relocations, waiting times and unproductive hours can be kept to a minimum. A bad example in this connection is if the testers have to be physically too far removed from each other and the rest of the project. An adequate setup of the workplaces also has a positive influence on the quality of the test process. This concerns, for example, the quality of both the internal and external communication and the motivation and productivity of the people involved.

Tips

- Find out at as early a stage as possible what the waiting times are in respect of the various requirements
- Ensure that any relocations, etc., are separately budgeted
- If testers are physically far removed from each other, extra hours for overheads may possibly be budgeted. This will make the disadvantages of the chosen office infrastructure clearer.

4) Establishing infrastructure planning

The test manager documents the agreements made and creates a general plan containing the timings of the availability of the various facilities. The further ordering and arranging of the infrastructure comes under the responsibility of the test infrastructure coordinator.

Products

The description of the necessary infrastructure, including a planning, established in the test plan.

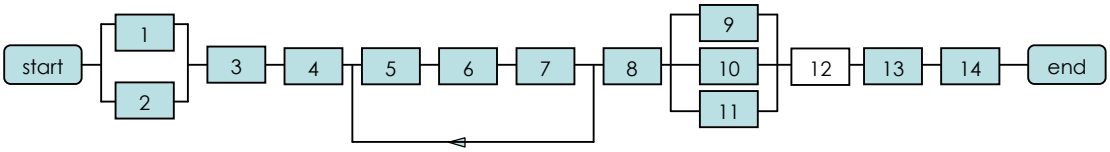
Techniques

Checklist "Test environments" (www.tmap.net)
Checklist "Office setup" (www.tmap.net).

Tools

Not applicable.

4.3.2.12 Organizing the management



Aim

To establish the way in which the management of the test process, infrastructure, test products and defects is organized.

Method of operation

The method of operation covers the following subactivities:
1. Defining test process management
2. Defining infrastructure management
3. Defining test product management
4. Defining defects management.

At test plan level, norms and standards can be set up for this, supported by procedures, templates and tools (test management tools, plans and progress monitoring tools). Sometimes at the overall level facilities are arranged to be used.

1) Defining test process management

Test process management is aimed at administering the test process in terms of progress and quality, and providing insight into the quality of the test object. To this end, identification, registration, administration, storage and interpretation of the following details has to take place:

- Progress and the expenditure of budget and time
- Quality indicators
- Test statistics

This management is sometimes assigned to a dedicated role: test project administrator.

This information forms the basis for managing and reporting by test management. Since control over the test process is increasing in importance, management is under pressure regarding the test process. Fast – preferably real-time – insight is required into the actual status quo. In this connection, the term **dashboard** is used: a simple overview from which all the superfluous information is removed and that provides the most important

information at a glance: the quality of the test object (in terms of defects) and the progress of the test process. Planning and progress monitoring tools but also testware management tools can be an excellent support here.

Below is an example:

	Regression	Subsys1	Subsys2	Subsys3
User Stories (test basis)				
Total	n.a.	103	23	n.a.
Status 1 (inferred)	n.a.	62	23	n.a.
Status >= 2 (part of testing)	n.a.	41	0	n.a.
Manual test scripts				
Total planned	291	145	35	111
Ready	291	62	0	93
To be amended	0	63	14	18
To be made	0	20	21	0
Results of last manual test round				
Date	n.a.	n.a.	n.a.	n.a.
Total run	n.a.	n.a.	n.a.	n.a.
OK	n.a.	n.a.	n.a.	n.a.
Not OK	n.a.	n.a.	n.a.	n.a.
Not run	n.a.	n.a.	n.a.	n.a.
Not completed	n.a.	n.a.	n.a.	n.a.
Automated test scripts				
Total planned	240	n.a.	n.a.	111
Ready	180	n.a.	n.a.	1
To be amended	180	n.a.	n.a.	0
To be made	60	n.a.	n.a.	110
Results last automated test round				
Date	18-12-05	n.a.	n.a.	17-12-05
Total run	111	n.a.	n.a.	1
OK	43	n.a.	n.a.	1
Not OK	66	n.a.	n.a.	0
Not run	2	n.a.	n.a.	0
Not completed	0	n.a.	n.a.	0
Defects				
New	9	13	n.a.	0
Open	197	1	n.a.	22
Being solved	20	5	n.a.	0
To be retested	14	0	n.a.	1
Closed	274	5	n.a.	143

(n.a. = not applicable)

Progress and expenditure of budget and time

The progress information offers the client and the test management insight into the test process. On the basis of this, the test process can be redirected, if necessary. Where there are negative trends, timely measures can be adopted.

The parts to be managed are the activities and/or products, related to hours, resources, timeline and with mutual dependencies.

In more detail

Most activities result in one or more products, such as (master) test plan, reports, test scripts, test files, test logs, etc. Exceptions are supporting activities, which usually do not deliver any tangible products. A choice has to be made as to whether to register the progress at the level of activities or at the level of products, with the further possibility of the mix form. The advantage of managing at the level of products is that these are easier to measure than activities: it is easier to judge whether a product is 80% ready than an activity, and more and more development and project management methods manage on the basis of products. With the identification of
activities or products, attention must be paid to the required degree of detail. Is it important to register an activity of several hours separately, or is it more efficient to register this as a part of a bigger activity? This is determined in consultation with the client.

Quality indicators

The aim of testing is to provide information and advice on the risks and quality of the object to be tested. To be able to provide this information, quality indicators are registered. The best-known and most obvious indicator is the defect. By establishing all kinds of details on a defect, such as e.g. status, severity, cause, quality characteristic and system part, all kinds of qualitative information can be gleaned from the defects at a later stage. Bear in mind the number of open defects relating to a particular part of the system, the number of defects found in a particular period, the number of defects relating to the requirements, etc. For more information on defects, refer to section 4.7 "Defects management". Various other indicators are also possible. For example, the number of retests or the number of breakdowns within the test infrastructure (as an indicator of its reliability).
The above-mentioned indicators tell us something about the quality of the test object. Another group of indicators tells us something about the quality of the test process itself:

Effectiveness of testing	Are the (important) defects being found?
Efficiency of testing	Are the defects being found as quickly and cheaply as possible?
Checkability of testing	Is the test process progressing transparently and in the agreed way?

Test statistics

The test manager builds statistics based on the above information. Statistics can supply insight into the progress of the test process and quality of the test object, including any trends. And statistics can also apply to the quality of the test process itself.

Tip

- The establishment of which data are measurable (metrics) is extensively described in section 4.11 "Metrics".
- Where a Testing line organization exists, it is advisable to confer with it on the statistics to be maintained. The line organization can possibly supply information as regards which statistics are important within the organization, and can possibly offer support. Correspondingly, the line organization is likely to be interested in the statistics from within the project.

2) Defining infrastructure management

The test infrastructure is subdivided into three groups of facilities:

- Test environment
- Test tools
- Office setup.

The test infrastructure is specified and ordered during the early stages of the test process. After installation, intake and acceptance of it, the infrastructure has to be managed. In practice, the management is usually transferred to a department, such as system management or operations, whether or not the test infrastructure coordinator forms the communication channel between the test process and the managing department.

In more detail

With regards to how to assign these management tasks, the various aspects of the test infrastructure can be divided into two groups:

- Technical management
  - test environment (hardware and software; management procedures)
  - test files (physical)
  - networks for test environment and office setup
  - technical office setup
  - test tools

  The most important tasks are:
  - Version management
  - Configuration management
  - Solving problem areas
  - Making logging available
  - Backup & restore
  - Recovery
  - (Technical) monitoring
  - Issuing authorizations
  - Providing availability
  - Implementing changes
  - Maintenance
  - Dealing with breakdowns.

  The technical management tasks that have to be carried out belong to the role of test infrastructure coordinator. With the execution of these tasks, support is given as required by the supplier or a department, such as system management or infrastructure services.

- Logistical management
  - the non-technical part of the office setup, such as canteen provisions, transport, entry passes, etc.

The tasks in the context of logistical management are not test-specific and as such are not discussed further in this workbook.

3) Defining test product management

At test-plan level, norms and standards are set up for the management of the test products, supported by procedures, templates and tools. This promotes the reusability of the products and communication on it. It is advisable to adopt the norms and standards generally applied within the system development process to documentation and configuration management. Test product management is sometimes assigned to a dedicated role: testware administrator.

The following are the various product groups to be managed:

- Products such as testware and test-project documents. Generally, higher requirements are set in respect of the management of reusable products like testware, e.g. that versions are retained.
- External products, such as the test basis and the test object. Responsibility for the management of this lies outside of the test process. However, the importance of good (version) management is extremely important to the test process. For that reason, requirements are often set from within the test plan in respect of the external management – e.g. that each product should be uniquely identifiable.

A choice has to be made as to which products are to be managed and to what degree. The management can be effectively supported by means of testware management tools.

In more detail

Below is a kick-start to a test-product management procedure. The procedure consists of four steps:

Delivery

The products to be managed are delivered by the testers to the manager. Preferably, the delivered files are placed in a separate directory. The products should be delivered complete (among other things, supplied with a version date and version number). The manager checks for completeness. The following are some of the items that could be checked:

- Name of author
- Type of document (also in document name)
- The definitive version number and version date
- Accuracy of references to other documentation (the test products should refer clearly to the associated test object and test basis)
- Mutations overview: overview of the versions, version dates and reason for change, including the name of the person who made the change
- Products in electronic form should be delivered with a fixed nomenclature, in a form that includes the version number.

Registration

The manager registers the delivered products in his administration on the basis of supplier's name, name of product, date and version number. At the same time, it is registered how long the relevant products should be kept. In certain cases, it may also be necessary to include the information on products related to the product to be registered. We find this in organizations where traceability is an important issue, for example because of legal obligations. With the registration of changed products, the manager should ensure that the consistency between the various products is preserved.

Archiving
A distinction is made between new and changed products. Stated roughly, new products are added to the archive and changed products replace the previous version.

Consultation
The issue of products to project team members or third parties takes place by means of a copy of the requested products. The manager registers which version of the products has been issued to whom and when.

In more detail

Traceability
Partly because of legislation (IFRS, SOX, FDA (Food and Drug Administration) and FAA (Federal Aviation Administration)), it is becoming increasingly important to demonstrate both that testing is being carried out and also what exactly is being tested. Showing what is being tested is achieved through traceability (demonstrating which test cases bear a relation to which part of the test basis). The proof that testing is actually being performed has to be supplied through explicit reporting. A subsequent requirement is to provide proof that the defects have been dealt with. If these stringent requirements concerning traceability and submission of proof are to be met, then the test product management, defects management and quality assurance in respect of testing should be tailored to this end (and extra budget made available for it!). The test management should be set up in such a way that the traceability and evidence can be followed step by step. This means that:

- It is clearly indicated in the test specifications from which part of the test basis these are derived
- With the test execution, the evidence to be submitted relates to which test cases have actually been executed
- It is made apparent which test cases have led to which defects
- The evidence to be submitted is established during the retest; which defects have been solved and approved in a retest.

Apart from this, traceability has the following big advantages for testing:

- Much insight is gained into the quality and intensity of the test, because from the requirements, the functional and the technical design and the software, it is known with which test cases these have been checked (or will be). The chances of omissions in the test are therefore much reduced
- With changes in the test basis or the test object, it can be quickly deduced which test cases need to be amended and/or carried out anew
- If, owing to pressures of time, it is not possible to carry out all the planned tests, test cases will have to be scrapped. Because the relationship with requirements, specifications and software is known, we can scrap those test cases of which the associated requirement or specification presents the least risk in production and it is clear with which requirements or specifications no, or no well-founded, decision is possible on the quality.

If the test needs to provide traceability, then the deployment of tools for test product management is more or less indispensable.

4) Defining defect management

A defects procedure should be set up to facilitate the handling and managing of defects. Ideally, this procedure is supported by a tool. Since a defects procedure applies to the entire project and not to a separate test level, this procedure can best be defined at master test plan level. This also makes it possible to detect overall trends, over and above

test levels. A description of the defects procedure is included in section 4.7 “Defects management”. This management is sometimes assigned to a dedicated role: defects administrator.

Products

A description of the various management processes, established in the test plan.

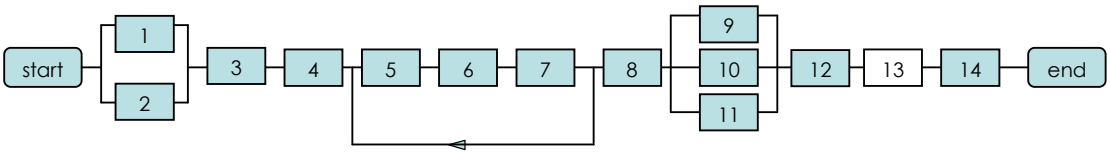
Techniques

Not applicable.

Tools

Defect management tool
Testware management tool
Planning and progress monitoring tool
Workflow tool

4.3.2.13 Determining test process risks (& countermeasures)



Aim

To cite explicitly the risks for the test level. This will provide the client and other stakeholders with a better understanding of the risks for the test, and they can allow for these in directing the total process.

Method of operation

In performing the preceding activities, the test manager has obtained a picture of the possibilities (and/or impossibilities) in connection with the test process, but also of threats and risks. In the test plan, an indication is provided per risk whether measures have been taken – and if so, which ones – to cover or reduce the risk found. Bear in mind here preventive measures for avoiding risks, but perhaps also measures to enable timely detection of problems. These risks are then monitored during the management of the test process.

It should be realized that this step is no more than paying mind to the risks as they are known *at the beginning* of the phase. Thereafter, the test manager includes these risks in the progress report under the separate section "Project risks". Subsequently, these risks are tracked, monitored, removed, new risks found, etc. If this activity takes place at project level, it can be combined with it.

In more detail

The risks can relate to, among other things:

- *Planning realism*
  The test plan depends on the plans of the various other parties. How realistic are these plans?
- *Entry quality*
  The two most important forms of input for the test process are the test basis and the test object. If this input is of insufficient quality, this will be very disruptive to the test process.
- *Resources*
  Testing requires people and means, in a certain quantity and of a certain quality. In practice, it often appears at the execution stage that the resources agreed in the plan cannot be (entirely) delivered in time.
- *Stability*
  To what extent will the test basis change during the test process? The more changes, the greater the consequences for the test process in terms of rework.
- *Infrastructure*
  Is it stable enough for the test; does the environment have to be shared with other parties; is the environment sufficiently representative; is enough support available? In many test projects, the infrastructure forms the most unmanageable risk.

Products

A description of the found (test) project risks and possible measures, established in the test plan.

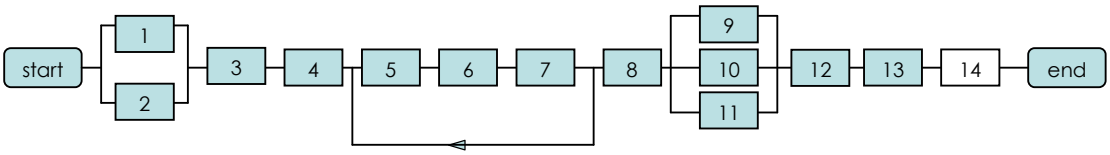
Techniques

Checklist “Test process risks” (www.tmap.net).

Tools

Not applicable.

4.3.2.14 Feedback and consolidation of the plan



Aim

To document the results of all the activities performed to date and obtain the client's approval of the chosen approach.

Method of operation

The method of operation covers the following subactivities:
1. Creating the test plan
2. Feedback on the test plan
3. Consolidating the test plan

1) Creating the test plan

The results of all the activities carried out so far are documented in the test plan. The test plan may for instance contain the following (commonly used) sections:

- Formulation of the assignment
- Test strategy
- Substantiation of the test strategy (test units, with the test varieties and techniques (approaches, coverage types, test design techniques) to be used per test unit)
- Organization
- Infrastructure
- Management
- Threats, (project) risks and measures
- Budget and planning
- Appendix: Product risk analysis

2) Feedback on the test plan

The various parts of the plan should be consistent. In practice, setting up a consistent plan takes place in several stages. The test plan with the results of preceding activities is fed back to the client and other stakeholders (such as the test manager of the overall test process) for approval or adjustment. This makes the test method of operation to be followed transparent and manageable, entirely in line with BDTM.

Tip

Some test managers have good experiences with going over the plan in a walkthrough session with the most important stakeholders. Any conflicts soon come to the fore, so that the number of feedback cycles can be kept to a minimum.

By adjusting the strategy (whereby the risk analysis is in principle unchanged), the test manager can enable the client to manage on the basis of the test effort weighed against the test intensity. This results in a suitably adjusted strategy, with the scrapping or adding of test intensity being shown by ○ or ● respectively, instead of ●. The test manager should make the consequences of this adjustment for the budget, planning and risks clear, and translate them into terms that the client will understand (referring back to the test goals).

This is repeated until the client is satisfied with the balance between test intensity and test effort.

Tip

A potential pitfall is that the communication on the adjusted strategy may be too "strong". If the client opts for a number of lighter tests than advised, a table is created that shows a lot of ○'s. If this table is shown repeatedly in progress reports or meetings, it gives two impressions: 1) the client is reckless, and 2) the test manager does not entirely approve and is distancing himself from the test method of operation. For that reason, it is advisable to use this table style only at the beginning and end of the test phase.

3) Consolidating the test plan

Following the feedback and possible adjustment of the plan, the test manager should submit the test plan to the client, at the least, for approval. Whoever else has to give their approval depends on the organization. In many organizations, the test plan is also submitted to other stakeholders for approval, such as users and developers. Parties for whom requirements are set in the assumptions part of the plan should give their approval.

Tips

- To make creating a test plan easier and prevent approval delays, it may be decided to have the test plan approved in parts
- The degree of formality of the approval depends on the organization. In some organizations, it is advisable to enforce the approval formally by having the test plan signed by the client and/or other stakeholders. In other organizations, the sending of approval by e-mail or a verbal confirmation will suffice.

The plan is then placed under configuration management as a formal test product. Besides this, a presentation, for example to the various stakeholders, can contribute to obtaining approval and – at least as important – create support throughout the organization.

Products

The test plan.

Techniques

Not applicable.

Tools

Not applicable.

4.3.3 Control phase

Aim

Providing the client with sufficient insight into, and the opportunity to influence, the following:
- The progress of the test process

- The quality and risks of the test object
- The quality of the test process
To this end, the test manager manages the test process optimally and reports on it.

Context

The activity referred to relates to the system test or acceptance test. The testing may relate to new build, maintenance, migration, a package implementation or a mix, and the development approach may be waterfall, iterative, agile or – again – a mix.

Preconditions

This activity begins after the creation of the test plan.

Method of operation

The test manager and the administrator(s) perform the activities assigned to them in the test plan. They manage the test process, the infrastructure and the test products. Using the data thus obtained as a basis, the test manager analyses possible trends. He also keeps in close touch with developments outside of the testing, such as any delays on the part of the developers, upcoming major change proposals and project corrections. If necessary, the test manager proposes particular measures to the client.
Information is the most important product of testing. Therefore, the test manager provides various types of reports to the different target groups, bearing in mind the BDTM elements of Result, Risks, Time and Costs.

Roles/responsibilities

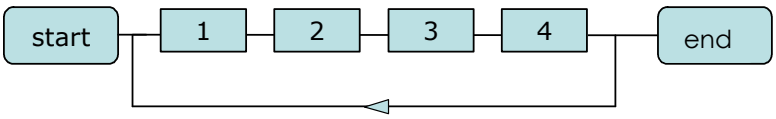
The test manager, also known as test coordinator, has primary responsibility for the management of the test process.

Activities

The control of the test process covers the following activities:

1. Management
2. Monitoring
3. Reporting
4. Adjusting

The scheme below shows the sequence and the dependencies between the various activities:



4.3.3.1 Management

Aim

Managing the test process, the defects and the test products with the goal of providing continuous insight into the progress and quality of the test process and the quality of the test object.

Method of operation

The management activity can be divided into two sub-activities:

- The following forms of management are carried out in accordance with procedures established in the test plan: the management of the test process, test product management and defect management. Infrastructure management is part of the "Setting up and Maintaining Test Infrastructure" phase.
- The test process is supported by – and checked for the application of – norms and standards.

These operations fall within the role of administrator or test manager. The following administrator roles for this are distinguished: test project administrator, testware administrator and defects administrator.

In more detail

The real challenge in management is not so much following procedures, but ensuring that the other test team members do so. Matters such as submitting timesheets, placing the testware under configuration management and carefully administering defects are not equally high in the popularity stakes among all testers. Measures for ensuring that this remains in focus are:

- "Repeat, repeat, repeat" the message that good management is crucial to the success of the test process. Make the reasons and advantages clear
- Make "management and control" a fixed subject in the periodic team meeting
- Remind people (directly) when they do not, or do not sufficiently, keep to the agreements
- Check activities and results, particularly at the beginning of the process to prevent bad habits from starting, and at the start of the test execution when the testers are working under time pressure.

N.B.: The test manager can take on the role of supervisor, or delegate it to another individual. Obviously, in the latter case the person should have the full support of the test manager when he admonishes someone for not complying with the procedures.

In practice, the setting up of norms and standards takes place concurrently with the development of the first products, so they do not exist at the time of writing the test plan. The supervisor will have the task of supporting the development of the first products and subsequently of creating generally applied templates.

Products

A managed test process.

Techniques

Not applicable.

Tools

Defects administration tool
Testware management tool
Workflow tool
Planning and progress monitoring tool

4.3.3.2 Monitoring

Aim

Monitoring the test process, based on internally managed data and external information.

Method of operation

The principal and most difficult task of the test manager is the monitoring of the execution of the plan.

While this is described in the section below as mainly an instrumental activity, it is equally a *communication activity*. The biggest part of the test manager's task perhaps consists of "monitoring" the employees on the team. This includes everything, from recruiting new testers during the testing process, delegating the work, holding work consultations/team meetings, supporting, coaching and assessing employees, up to and including the conducting of exit interviews. Another very important task for the test manager in this same connection is the maintaining of contact with the world surrounding the test team, also known as *stakeholder management* or *expectation management*. Do the expectations of the test clients still correspond with what the test is going to deliver? Are there developments in the project that will influence the test process? It should be obvious that highly developed social and communication skills would not go amiss here.

In general, the activities described in the plan – such as preparing, specifying and executing the tests – should be carried out according to a particular timeline and in a particular sequence. To do this, the test manager has the necessary people at his disposal (including himself). He sets out a detailed planning for the coming period, outlining who will do what, in how many hours. This is necessary, as the planning within the test plan is not detailed to the extent that tester A knows that, in the coming week, she should specify test units X and Z, and tester B knows that he is to carry out test scripts Y1 and Y2 for test unit Y. Experience shows that such detailed planning only works for the initial short period, after

which changes are always taking place, requiring the planning to be revised. Most obvious periods for which a detailed planning can be set up are the phases: Setting up and maintaining infrastructure, Preparation, Specification, Execution and Completion. With iterative system development, the test manager also makes a detailed planning per iteration. In setting up a detailed planning, the test manager makes allowance for all the aspects of planning, such as priorities, availability and skills.

Another of the test manager's tasks is to fill in "blank spaces" in the test plan during the course of the test process. This is the case when, at the time of setting up the plan, certain information is missing or there is no time to carry out a particular activity.

In more detail

For example, the allocation of test units and/or test techniques. Occasionally, information from the developers is lacking, so that it is not possible to arrive at a satisfactory distribution of test units. The test manager may also decide to delay the allocation of test techniques to test units until the testability review has been carried out.

Towards the end of the test execution, the monitoring becomes even more important, as the test manager must then be able to answer the question of whether stopping testing is justified. The exit criteria formulated in the plan are the deciding factors here, but if they are absent or no longer current, there are some rules of thumb available:

- Have all the planned tests been executed (in accordance with the latest test strategy)? This emphatically does not concern the original strategy, but the latest, amended version. This contains the most recent insights of the client and test manager into the balance between risk and test coverage
- Are the number and degree of severity of the outstanding defects at an acceptable level? And to this may be added: have the costs of the testing during this period risen higher than the returns ("damage prevention", see below)?
- Has the number of newly found defects as well as the number of solved and retested defects been reduced to a minimum during the latest period (e.g. week)? This last point says something about the stability of the system. Sometimes, in the last period, so much has been reworked and retested that the system has several releases per day. If only both of the above points were to be considered, it could be decided to stop testing. However, the system is still anything but stable, and a regression test is strongly to be advised.

Only when a positive answer can be given to these questions does it make sense to recommend ending the testing.

After the test team has completed all its tasks, including the Completion phase, the test manager asks the client to terminate the assignment and to discharge the test team. The team is then disbanded.

Tips

- Damage prevention: one of the benefits of testing is that in production costs do not arise because faults do not occur. This could possibly be conveyed by relating the severity and cause of a problem to any repair costs after going into production: what would the costs have been if the defect had not been found?
- It is possible to make a model with which the prevented damage of each defect can be estimated. In this, a certain factor is allocated to aspects of a defect (severity, cause and quality characteristic), e.g. a severe defect delivers on average 8 times the damage of a cosmetic defect. By estimating the prevented damage of a limited

number of defects with the aid of experts, the factors are determined and the average sum per defect. The prevented damage of a new defect can then be quickly estimated by multiplying the average amount by the relevant factors (see [Aalst, 1999])

- A simple but useful graphic is the S-curve of the cumulative number of found defects per day (see figure 48). Where the S starts to flatten out, this could be an indication for stopping the testing. In any case, it is an indication that it is time to discuss whether or not to stop with the stakeholders.

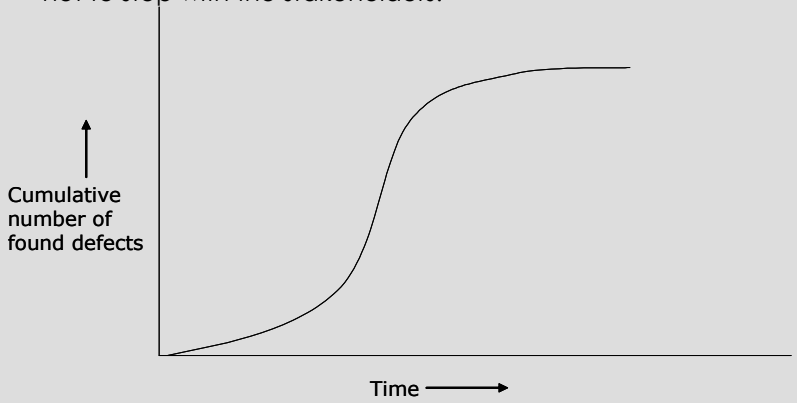


Figure 48. Example of S-curve.

Practice also teaches us that the original plan is bound to be amended. The amendments can have both internal and external causes, i.e. from both within and beyond the test process. It is up to the test manager to flag these events or trends as early as possible. Measures for redirecting a negative trend can then be adopted promptly. This is almost always better, cheaper and faster.

In more detail

While in practice there has probably never been a project where the plan was carried out unchanged, this does not mean that the plan is somehow unimportant. On the contrary, the plan provides a common framework that makes correcting the process easier and more effective than when working without a plan.

Information about the events or trends comes from the internally managed data and from outside the test process, e.g. minutes or memos, but not least also from verbal interchange, such as the project consultation, stand-up meetings, bilateral discussions, etc. This is where a good social (project) network shows its worth to the test manager. Using this information, the test manager analyses possible trends and tries to apprehend threats (or indeed opportunities) in time: will the trend continue? What needs to happen to prevent it?

For this purpose, the test manager carries out the following steps:
1. Analyzing the event, estimating risks and defining countermeasures
2. Coordinating with the client and other stakeholders (optional, depending on tolerance)

1) Analyzing the event, estimating risks and defining countermeasures

The test manager analyses the cause of the event and determines the consequences for the test process. He also examines the significance of the event explicitly in respect of the risks that are covered at this stage of the testing. Events can influence the testing positively

or negatively, and the test manager determines the possible countermeasures, depending on the timing of the event, the analysis and the consequences for the test process.

In more detail

Below are some examples of common events and their causes, consequences and countermeasures:

Event	*The test object is delivered later than planned, while the deadline for the test process remains the original date.*
Possible causes	The causes of this will usually lie in the stage preceding the testing phase. Likely causes are higher degrees of complexity than expected, differences of opinion or expansion of the scope.
Consequences for the test process	• There is less time for the execution of the test cases • There is more time for specifying test cases • The required means for the test execution, such as a representative test environment, do not need to be available until later • Etc.
Possible measures	• Extra test capacity is requested for the test execution • The test cases are described more comprehensively during the specification phase, making their execution simpler for inexperienced testers • Tests are carried out in parallel with each other • Allowing for the risk category, it is decided to reduce or skip certain test activities • It is decided to push certain test levels or test types together, so that they can be carried out collectively • The roof tile method in respect of the specification of test cases and execution, i.e. delivering smaller batches more often from development to testing • An increase in budget is requested • Etc.

Event	*The productivity of the employees is lower than expected.*
Possible causes	• The quality of the test basis is less than the advance estimate • The employees have on average less experience than was expected in estimating productivity • The test object is more complex than previously estimated, so that setting up the test cases is more difficult • Etc.
Consequences for the test process	The test activities take longer than planned.
Possible measures	• Employees are replaced by others with more experience • Extra capacity is requested from the client • The test method of operation is adjusted, so that less risky parts are given still less attention, and more time is made available for the risk-bearing parts • A decision is made not to process all the test cases extensively and, for example, only to create logical descriptions (this generally places higher demands on the tester who executes the test) • Etc.

Event	*Specifying the test cases takes more time than was planned.*
Possible causes	• The test basis is of less quality than was planned • The productivity of the employees is lower than was previously estimated • Etc.
Consequences for the test process	Overrun of test specification can mean that the test execution cannot start on time.
Possible measures	• Techniques are selected that will result in fewer test cases. This also means that there will be less test coverage and that the recognized risks will have less coverage • The logical test cases are not written out into physical test cases (this generally puts higher demands on the executor of the test) • Extra time or capacity is requested from the client • Extra support is provided by subject-matter experts or developers • Etc.

Event	*The test basis keeps changing.*
Possible causes	• The test basis is of lower quality than was planned • The scope of the project keeps increasing • There are differences of opinion in the project concerning the functionality to be delivered • Etc.
Consequences for the test process	• (During specification and execution) The test specifications need to be continually reviewed and are never completed • (During execution) Extra retesting is required continually
Possible measures	• The logical test cases are not written out into physical test cases (this generally puts higher demands on the test executor) • Exploratory testing as a technique in order to be less dependent on the test basis and also to put off the need for this as far as possible • Extra time or capacity is requested from the client • Stricter configuration and change management at project level, (obviously with involvement of testing) • Etc.

In more detail

Retesting
A specific part of the strategy is how to deal with retesting. Normally, a test delivers defects that are then reworked. A choice must then be made as regards retesting. For example, limited retesting can be carried out, focusing only on the adjustment. Another possibility is to carry out retesting of the total function in which the adjustment was implemented, of the total function in conjunction with surrounding functions, or even of the total system. The change can also be retested with specific test cases and a regression test can be run on the (unchanged) rest of the system. The choice of the degree of retesting is made based on the risks. Sometimes guidelines are in place; sometimes the test manager determines the retesting level from case to case. In fact, the test manager takes a kind of mini test-strategy decision, with all the steps being gone through briefly.

2) Coordinating with the client and other stakeholders (optional, depending on tolerance)

Depending on the measures to be carried out, the test manager can carry them out independently, or prior agreement with the client and possibly other stakeholders may be necessary. The form that this coordination takes depends on the organization. In practice, use is often made here of reports.

The margin that the test manager has for taking measures independently is determined by the following factors:

- The degree to which the difficulties of the test process can be solved within the set assignment, the product risk analysis, the test strategy, budget, planning and other preconditions. In other words: the degree to which the client is to be left out of it
- The degree to which the difficulties can be solved within the limits of tolerance, which were agreed in the planning phase.

In practice, the test manager should generally ask permission if the measures would influence the agreements that were made at the planning phase. In other words, if adjustments have to be made to the formulation of the assignment, product risk analysis, test strategy, budget and/or plan.

In more detail

The devil's quadrangle

A familiar trend is symbolized in the 'devil's quadrangle', with Time, Money, Functionality and Quality as the corner points. At the start of the project, there is a certain balance between the points. A predictable course of events is that all kinds of unforeseen events occur that introduce tension into the quadrangle (see figure 49). In particular, certain activities overrun (Time) and/or cost much more than was estimated (Money). The project manager corrects this by putting restrictions on the other corner points, i.e. Quality and Functionality.

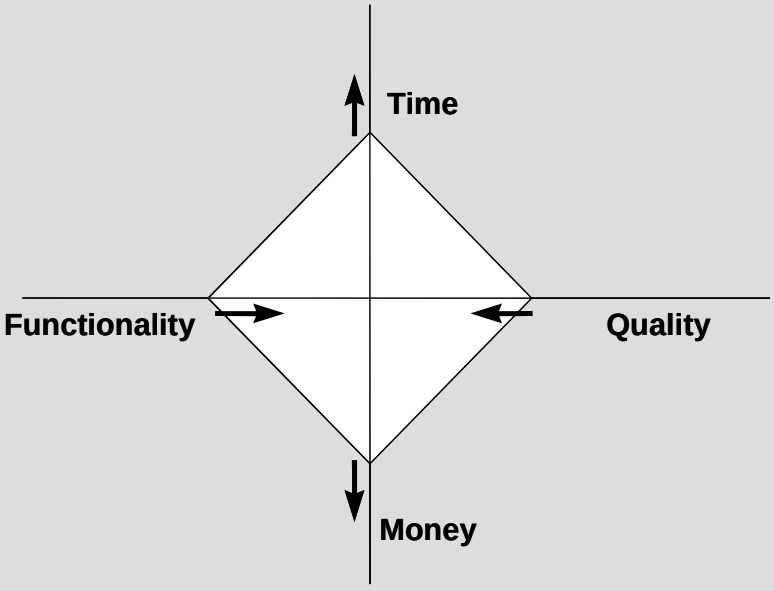


Figure 49. The devil's quadrangle.

Although in itself this is not necessarily "wrong" behaviour on the part of the project manager, it is the test manager's job to monitor Functionality and Quality. Bearing the

quadrangle in mind, the test manager indicates the consequences of the project manager's decisions and alerts the client, for example, if the choices repeatedly fall on restricting Quality and Functionality. Timely communication of this trend in particular is difficult, which emphases the importance of an independent test manager. Depending on their perceptions, the test manager is either the "conscience" or the "thorn in the side" of the project manager and/or client. This role requires a high degree of professionalism, for the test manager has to tread carefully regarding the politics of the various interests within and beyond the project.

Tips

- A tip that can be given in the above context is, when changes are requested or when the project manager proposes adjustments to the testing, never immediately to dig one's heels in and cry "No, not possible, because …" It is better to respond in the manner of, "Hmm, interesting idea. Let's tease that out a bit further; what would it mean in relation to… The idea could work if we all accept these and those consequences."
- At the Preparation and Specification phases, the (project) management tends towards indifference in respect of the testing. Only at the test execution stage, at the point when the testing is on the critical path, is interest shown in the progress. The test manager should make allowance for this, by, for example, adjusting the form and frequency of reporting and consulting (more of it, and more frequently, during the test execution) and, in the process, might well transform his image (from "walk-on part" to "star").

Products

Proposed management measures.

Techniques

Not applicable.

Tools

Defects management tool
Testware management tool
Workflow tool
Planning and progress-monitoring tool

4.3.3.3 Reporting

Aim

Creating reports that provide insight into both the quality of the test object and the progress and quality of the test process. These reports will ensure that the client and other stakeholders can steer the course of the testing effectively.

Method of operation

During the test process, the test manager compiles various reports. The form and frequency of reporting is established in the "Management" chapter of the test plan. Periodic reports are created on the quality of the test object and the progress and quality of the test process. Besides the periodic reports, the client or other stakeholders may request reports on demand. The most familiar example of this is the risk report, for outlining the possible consequences of a threat or risk to the testing. There may also be an unexpected request for an extra progress report, for example to provide the most up-to-date input for a steering group or project management meeting. At the end of the test process, a recommendation for release and final report are drawn up.

With all this information, the client, project manager and other stakeholders are supplied with insight into the extent to which:
- The intended result is achieved
- The risks of taking the system into production are known and are as small as possible within the set preconditions
- This has taken place within budget and term.

In other words, this refers to the BDTM aspects of Result, Risks, Time and Costs. Supplying insight implies that the report should have relevance to the recipient(s) of it.

The reports are based on the data as established in accordance with the section on "Organizing the management".

Tips

- Always report accurately and completely; it is in nobody's interest to present matters in an exaggerated light
- Report with precision and substantiate with reliable figures
- Report in the terminology of the client, not only in numbers of defects
- Report positive news, too, for example the number of test cases that have been processed without defects
- Regarding the level of detail in the report, answer the needs of the target group
- Be neutral in the wording; don't get personal
- Respond to questions like "Can I go into production?" or "Can it be accepted?" preferably not with "No!", but with "Yes, provided that ..." or "Not unless …"

The content of the most important reports is described below:
a. Progress report
b. Risk report
c. Release advice
d. Final report

a) Progress report

Reporting takes place in accordance with the reporting structure described in the test plan. The progress report contains data on the most recent reporting period and cumulative data on the entire test process.

Besides figures, the report should also provide textual explanation and advice on the results, progress, risks and any problem areas. The latter is inclined to be forgotten in reports that are generated from test-management tools. It should be realized that explanation and advice are very important in the provision of quick and reliable insight into the figures. It is the most important product of testing. While the explanation can and should be given verbally, it most definitely should be contained in the written report. This forces the test manager to think carefully, as well as making the advice stronger, reaching a wider audience and helping with the process evaluation in retrospect.

In more detail

Progress report versus final report
Although the terms 'interim report' or 'progress report' may suggest that these are less important than the final report, in fact the opposite is true. The progress report supplies early information and advice, with which the recipients (such as client, project manager and others) can often make timely adjustments for keeping the total process on the right track. The final report is more a retrospective evaluation that mainly benefits subsequent test processes and projects.

In outline, a progress report has the following content (based on the BDTM method with the four aspects of Result, Risks, Time and Costs). In practice, the list of contents may follow a different sequence; subjects may be combined, or even omitted. It depends on the report's target group.

1. Status of the test object (BDTM: Result)
    - 1.1 Status per characteristic/object part
    - 1.2 Status of test goals
    - 1.3 Trends and recommendations
2. Product risk and strategy adjustment (BDTM: Risks)
3. Progress of the test process (BDTM: Time and Costs)
    - 3.1 Progress (in hours and data) of activities or products over the recent period
    - 3.2 Activities in the coming period
    - 3.3 Hours lost
    - 3.4 Trends and recommendations
4. Problem areas/points of discussion (all the BDTM aspects)
5. Agreements
6. Quality of the test process (optional, all the BDTM aspects)
    - 6.1 Effectiveness
    - 6.2 Efficiency
    - 6.3 Verifiability

These subjects are further explained below.

1. Status of the test object (Result)

1.1 Status per characteristic/object part

It is shown per characteristic/object part:

- The status of the tests (not started, planned, specification, execution, retest X, completed), optionally with the progress percentage, e.g. the progress of the execution is estimated at 60%
- Overview of numbers of defects (sorted by status and severity, optionally also by other aspects, such as cause)
- If test products (such as test cases or test scripts) are seen emphatically as results, they can also be included in the overview, with an indication of whether a start has been made on the product and whether it is ready.

The closer the end of the test period approaches, the more attention is paid in the progress report to the consequences of open defects. In the beginning, it is less useful to include this in the report, since it is expected that the defects will be solved. But the consequences should always be included in the defect report itself.

- Defects that remain open and their impact
- Defects not solved (known errors), and their impact.

1.2 Status of test goals

Based on the above, the status per test goal (user requirement, business process, critical success factor, etc.) is reported. Sometimes a test goal can be directly linked to a number of characteristics/object parts and to the test status related to them; sometimes the status per characteristic/object part is not sufficiently usable and the test manager still has to determine the test status per test goal. The risk tables from the Product Risk Analysis make the link possible.

1.3 Trends and recommendations

Relevant trends and related recommendations can be reported here.

In more detail

<u>Below are some overviews that will reveal whether certain trends are taking place:</u>

- The number of open defects per week will indicate whether the testing can tail off or if a backlog is building up
- The relationship between numbers of defects and test cases per subsystem provides an indication of whether extra testing on that part will deliver many more defects
- The number of found defects and number of solved (including retested) defects within a certain period says something about the stability of the system
- Status of the defect versus who should carry out the following step in the handling of it. This shows up where any bottleneck lies. For example, where all the complex faults are allocated to that one experienced developer, with the result that a backlog of unsolved defects is created
- Cause of defects (requirements, design, code, test environment, wrong installation/operation, wrong test case) versus subsystem. Provides insight into the concentrations of specific mistakes
- Number of defects versus tables (with data warehousing). This tells us what the error-sensitive system parts are.

In more detail

In order to give the trend significance for the stakeholders, it is advisable to use graphics, making the trend visible. This is not as easy as it seems. It is difficult to produce a clear and legible graphic. A few tips (quoted freely from [Tufte, 2001]):

1. Make the data and the message the centerpiece
2. Maximize the data/ink ratio (i.e. leave out all the symbols, lines and colors that don't add anything)
3. Remove redundancies
4. Review and amend.

2. Product risk and strategy adjustment (Risks)

In this part of the report, the stakeholders are given insight into the degree to which the coverage of the various product risks has changed, as well as into any process risks.

In the test plan strategy, it is determined whether and to what degree product risks will be covered by testing. During the test process, aberrations may occur: the estimate of the risk appears different and/or the test coverage requires adjusting. The adjustments over the reporting period, with associated consequences, are reported in this part. In this, the translation is made into the test goals: what kind of impact will the changed risks have on the attainment of these goals?

3. Progress of the test process (Time and Costs)

Regarding the progress of the test process, the points below are significant.

3.1 Progress over the latest period

At the level of phases and/or main products, the following could be reported:

– Number of planned hours
– Number of hours spent so far
– Number of hours expected still to be spent
– Percentage completed
– Dates: planned/expected/actual start date; planned/expected/actual end date.

Products could be the test plan, test scripts, test-execution files and reports.

If the test manager is responsible for the budget, he will also include in the progress report information on completing the test process within budget.

3.2 Activities in the coming period

Here, the activities to be carried out in the coming period are reported.

3.3 Hours lost

This refers to non-productive hours of the testers. If the test process environment does not meet certain preconditions, this will result in inefficiency and loss of hours. Examples are a non-functioning test infrastructure, much or lengthy test-obstructing defects or lack of support. Hours lost, and the causes, are reported here.

3.4 Trends and recommendations

As with trends in the status of the test object, trends and recommendations in connection with the progress of the testing should also be reported. The central question here is whether the agreed milestones are (or appear to be) feasible.

In more detail

One of the trends that can be watched is the average time required for the reworking of a defect. If this increases, it is possibly a signal that the volume of the backlog of work is increasing sharply. The percentage of wrongly reworked defects can also be observed.

4. Problem areas/discussion points (all the BDTM aspects)

In this section of the report, the test manager points out any problem areas or points for discussion that jeopardize completion of the test assignment within the set limits of time and costs. For example:

- The test object being delivered later than planned
- The quality of the test basis being less than expected
- The test environment not being available on the agreed date
- Test-obstructing defects present in the test environment or test object.

Besides the various problem areas, their consequences and possible measures are shown. Here, too, the test manager makes the translation into the test goals.

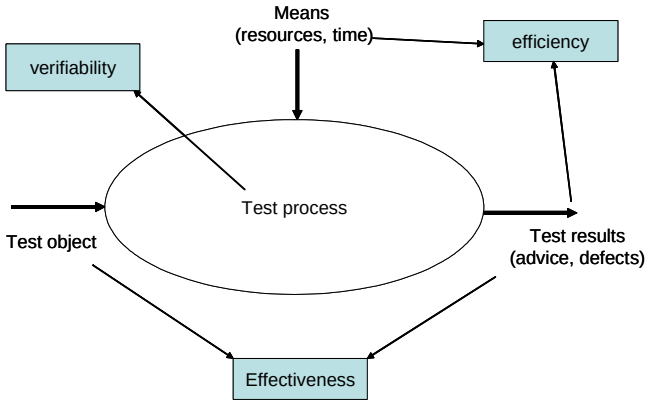
5. Agreements

This part shows the agreements made in the current period between the test team and other parties that are relevant to the recipients of the report.

6. Quality of the test process (optional)

If required, this part of the report can include information on the quality of the test process. The following questions play a part here (see figure 50):

- Are the significant defects being found (as early as possible)? (Effectiveness)
- How economical is the test process with time and resources? (Efficiency)
- Is the test process working as agreed? (Verifiability)



–

Figure 50. The three quality aspects of the test process

In more detail

A point of focus here is the general problem with metrics: how to draw the right conclusions from the figures; how to avoid comparing apples and oranges. See also section 4.11 “Metrics”.

Effectiveness

In more detail

The difficulty with the question of whether the testing is effective, is that this can usually only be established in retrospect. The effectiveness issue can be split into two parts:
– Is there a good strategy in place?
– Is the testing being carried out in accordance with this strategy?

There are various indicators that can be included in the report:
- The percentage of found defects in the test level / the number of defects present or an approximation thereof; the number of defects can be approximated by, for example, the number of defects still being found during the first 3 months of production
- The percentage of found defects in a test level that should reasonably have been found in a preceding test (30% of the defects found in the acceptance test concern programming defects; these should actually already have been found in the development tests and system test)
- Degree of testing coverage; the more thorough the test, the more defects will be found
- The percentage of mistakes (= test faults).

Efficiency

In more detail

The following are possible indicators of this:
- The number of defects found per test hour
- Estimating prevented damage in relation to the test costs (through finding faults)
- Number of specified or executed test cases per hour
- Number of reviewed pages per hour.

By comparing these figures with an established standard, a picture is created of the efficiency of the test process.

Verifiability

In more detail

This aspect is difficult to communicate through indicators. What the test manager can say in the report about this is whether and how in the latest period it was verified that the test team was working as agreed. The verification can focus on the test products or the processes and can be based on the planned quality measures, or on monitoring, or on a random check at the overall level. The test manager should make a good risk estimate as regards what checking would be useful. In particular, the test levels that are placed with inexperienced test managers or that have been outsourced are eligible for verification.

Below is an example of a dashboard, enabling the most important information to be seen at a glance.

Part	Is	Was	Remarks
1. Quality of test object	☹	😐	…
2. Risks	😐	😐	…
3. Progress	☺	☺	…
4 Quality of test process	☺	😐	…

Later in the report, these points are worked out in detail in overviews with notes. Examples of overviews (without notes):

Quality of test object - defects

	Open	To be retested	Closed	TOTAL
Obstructing			4	*4*
Severe	1	1	12	*14*
Disruptive	3	12	49	*64*
Cosmetic	15	7	37	*59*
TOTAL	*19*	*20*	*102*	*141*

Quality of test object – subsystem x causes

	Require-ments	Design	Software	Infra-structure	Test	TOTAL
Subsystem 1	3	5	18	6	2	*34*
Subsystem 2		1	16	2	4	*23*
Subsystem 3	6	14	30	14	1	*65*
Total system						
TOTAL	9	*20*	*64*	*22*	*7*	*122*

Progress

	Time			Hours			
Milestone/Activity	Plan	Expected	Realized	Estimate (A)	Spent (B)	To be spent (C)	Diff. (A-B-C)
Planning							
- Test plan	Apr 1		Apr 1	60	54	0	6
Preparation							
- Detailed intake	Apr 8	Apr 12		40	46	4	-10
Specification							
- Test unit 1	May 2	May 2		120	60	60	0
…							

b) Risk report

The purpose of the risk report is to supply the various stakeholders with sufficient information to allow them to make informed decisions in respect of the test process. The information in the risk report should therefore also focus on the consequences of the event for the achievement of the agreed result within the agreed timeline and cost levels.

In more detail

The test manager creates a risk report if events take place for which measures are required to be taken that the test manager is not authorized to decide upon.
Another reason for creating a risk report is if the client asks the test manager to set out consequences and possible measures for one or more scenarios upon which a decision is required to be taken. For example, a scenario in which the client sees that the development activity is overrunning and he considers making budget available from the test.

In a risk report, at least the following subjects are dealt with:
- A description of the event / the scenario
- The consequences of the event for the testing
- The significance of the event to the degree to which the various product risks are covered

- Possible countermeasures
- If possible, the test manager outlines several measures with the associated costs. An estimate is also made of the influence of the measures on the recognized consequences and degree of coverage of product risks
- Recommendation
  The test manager provides a recommendation in respect of the measure(s) to be selected.

c) Release advice

The release advice is created at the end of the test execution. The purpose of the release advice is to provide the client and other stakeholders with a level of insight into the quality of the test object that will allow them to make informed decisions on whether the test object can go on the following stage with its present status. The following phase in this connection refers to a subsequent test level, or production. For that reason, the release advice is usually created under severe pressure of time, since immediately after execution of the last tests and before the test object is released to the next phase, there is usually very little time available. The test manager would do well to have a draft release advice largely prepared towards the end of the last tests, so that only the last test results need to be processed.

The information in the release advice should not actually come as a surprise to the client. He has been kept abreast of developments relevant to him by means of reliable progress reports and, where necessary, risk reports.
In order to supply the client with the information necessary at this stage, the release advice should cover at least the following subjects:

- A recommendation as to whether, from the point of view of the testing, it would be advisable to transfer the test object in its present state to the next phase
  The final decision on whether or not to go on to the next phase does not lie within the test process. Many more factors are at work here, other than those relating to the test process. For example, political or commercial interests that make it impossible to postpone transfer to a subsequent phase, despite a negative release advice

- Obtained and unobtained results
  Which test goals have been achieved and which not, or only to a certain degree? On the basis of test results on characteristics and object parts, the test manager gives his opinion and advice on the test goals set by the client. It is also indicated whether the exit criteria have been met. The number and severity of the open defects play an important role here. Per defect, it is indicated what the consequences are for the organization. If possible, risk-reducing measures are also indicated, such as, for example, a workaround, allowing the test object to go on to the next phase without the defect being solved.

- Risk estimate
  During the planning phase at the beginning of the test process, an agreement is made with the client about the extent to which product risks will be covered, and with what degree of thoroughness. For various reasons, it may be decided to cover certain parts less thoroughly with testing than the risk estimate indicates. Moreover, during the test process, all kinds of changes are still usually being made to the original strategy; moreover, the original risk estimate has possibly been adjusted, perhaps resulting in additional or different risks. In this part of the release advice, the test manager points out which characteristics or object parts have not been tested, or have been less thoroughly tested than the risks justify and so present a higher risk. The associated consequences are also shown.

d) Final report

The purpose of this report is to obtain insight into the way the test process has gone and to document empirical data for the purposes of future test processes.

The final report is created after issuing the release advice, usually when the test object has already been released to the next phase. More time is therefore available for it.

The contents list of a final report is more or less the same as that of a progress report:

1. Evaluation of the test object (BDTM: Result)
    - 1.1 Status per characteristic/object part
    - 1.2 Status of test goals
2. Product risk and strategy adjustment (BDTM: Risks)
3. Release advice (BDTM: Result, Risks)
4. Evaluation of the test process
    - 4.1 Progress (BDTM: Time and Costs)
    - 4.2 Quality of the test process (BDTM: all the aspects)
6. Recommendations for future tests
7. Empirical data (optional)
8. Costs/benefits analysis (optional)

However, whereas a progress report looks ahead, the final report looks back. In other words, it mainly concerns the difference between the original plan and the final realization. What degree of deviation is there from the original plan? Was the plan a good one, or were issues wrongly estimated? Were adjustments always timely and effective? To what extent were the preconditions met, and met promptly enough? Could bottlenecks have been prevented? These differences are analyzed in particular for purposes of the risk analysis, test strategy, estimate and planning. The quality of the test process is also considered: were the chosen procedures, tools and techniques used correctly and was the test environment satisfactory? Recommendations are provided, if possible, for future tests. The activity 'Evaluate the test process" (see section 6.8.1) supplies the input for this evaluation. Also, use can be made of the "Test process evaluation" checklist. In addition, empirical data may be collected and made available to the client, or, even better, to a Testing line organization. A last, optional, part of the final report is a costs/benefits analysis.

The final report is made available to the client and other stakeholders, possibly by means of a presentation.

In more detail

Empirical data

Examples of empirical data are:

- Size of the test object
- Development effort
- Number of defects
- Duration and hours per main activity
- Duration and hours required for specifying tests
- Duration and hours required to execute the tests
- Number of test cases
- Analysis of lead time per defect
- Number of defects to be expected
- Number of retests.

A comprehensive summary of the empirical data that can be collected is included in the list "Metrics list". That section (4.11) also discusses the Goal-Question-Metric method for implementing metrics

Costs/benefits analysis
The costs of the test process are relatively simple to establish. Bear in mind, for example, the costs of the used resources, manpower and equipment. The benefits of the test process, however, are more difficult to establish. As indicated in the section "Why test", there are four types of benefits of testing. It is difficult, but not impossible, to provide a quantitative indication of these.

Products

Reports (progress report, risk report, release advice, final report)
Empirical data
Costs/benefits analysis

Techniques

Checklist "Test process evaluation" (www.tmap.net)

Tools

Defects administration
Testware management tool
Workflow tool
Planning and progress monitoring tools

4.3.3.4 Adjusting

Aim

Adjusting the test process (in consultation with the client as necessary).

Method of operation

When the proposed measures have been reported, the client has agreed and a selection has been made from one or more of the possible alternatives, the test manager can put them into effect. To this end, he carries out the following steps:

1. Implementing measures and evaluating effectiveness
2. Adjusting products from the planning phase (optional, dependent on tolerance)
3. Feedback to the client

1) Implementing measures and evaluating effectiveness

In this step, the test manager implements the (approved) measures. After some time, he assesses whether the desired effect has been reached with the adopted measures.

2) Adjusting products from the planning phase (optional, dependent on tolerance)

The measures can have consequences for the agreements as set out in the test plan. In that case, the test manager adjusts the products concerned and submits them to the stakeholders for approval.

Examples of adjustments to the various products are:

- The scope of the assignment is adjusted. This is the case, for example, if it is decided to carry out one or more extra test types, or to omit them
- The product risk analysis is revised, because during the execution of the test process it appears that the probability of faults was wrongly estimated. This is the case if the development tests were limited because of pressures of time
- During the test execution, changes will be made in particular to the test strategy if the test intensity or method of operation is amended. For example: under pressure of time, it is decided to create no more test cases for the testing of screens, but to use a checklist
- Many of the events mentioned in the section on "Monitoring" have consequences for the budget. A common example of such an event is delay in delivery of the test object while the deadline for the test remains unchanged. The planned coverage is then only feasible if extra people are brought in, resulting in lost time (initiation) and management overhead.

Since the formulation of the assignment, product risk analysis, test strategy and estimate are required to be consistent with each other, a change in one of the products will usually lead to changes in the other products. Changes to the test plan are established in a new version or in a supplement, which is again submitted to the client for approval. It is the test manager's responsibility to communicate clearly to the client the consequences of the changes.

3) Feedback to the client

In this step, the test manager reports to the stakeholders, such as the client, on the measures taken and their consequences for the test process. If the client (and possibly other stakeholders) were involved earlier in giving permission to adopt the measure, this report will generally contain no new information. Even if the test manager is able to

implement the measure independently, the event and associated measures are reported to the stakeholders to keep them abreast of the testing developments. The periodic progress report is a suitable means for this.

Products

Steering measures
Amended plan

Techniques

Not applicable.

Tools

Workflow tool
Planning and progress monitoring tool.

4.3.4 Setting up and maintaining infrastructure phase

Aim

To provide the required test infrastructure, which is used in the various TMap phases and activities.

Context

The test infrastructure consists of the facilities and resources necessary to carry out the testing satisfactorily. A distinction is made between the facilities for test execution (test environments), for supporting the testing (test tools) and for the day-to-day work of the testers (workplaces).

Definition
The test infrastructure consists of the facilities and resources necessary to facilitate the satisfactory execution of the test. A distinction is made between test environments, test tools and workplaces.

The setup and maintenance of infrastructure involves specific expertise. It is something that testers in general have limited knowledge of, but upon which they nevertheless are very dependent (without infrastructure, there can be no test). All the responsibilities surrounding the setting up and maintaining of infrastructure are therefore often given to a separate maintenance department, necessitating close co-operation with these other (sometimes external) parties during the test. This means that test managers land in a situation where they have no authority over the setup and maintenance of the infrastructure (the maintaining party has the say-so), while they nevertheless depend on it. This can lead to conflict. For example, the situation could arise in which this maintaining party gives priority to solving production-disrupting problems above solving problems in a test environment. Furthermore, a maintenance department often also has particular security guidelines (e.g. authorization checks, fixed backup times, installation procedures) that cannot easily be ignored. This is something that should be taken account of during the testing and, with that, the responsibility for the setup and maintenance of the infrastructure is an important

area of focus for the test manager. A means of alleviating the concern for this support process is the permanent test organization, which will take full responsibility for the setup and maintenance of the test infrastructure.

Example

An organization's infrastructure is maintained by an external party, with the condition that a daily backup of the infrastructure is made. For this purpose, an automated process is created that makes a backup at night, somewhere between the hours of 22:00 and 06:00, depending on other processes.

The building and testing of a new web application overruns and it is decided to extend the time spent on testing per day. This means that the testers plan to test (in shifts) from 06:00 to 01:00 hours. It is therefore necessary to change the times of the backup process. A request is submitted, but the external organization is reluctant to grant it. Many other processes will have to be changed, and that could take up to two weeks. The option of not making a backup of the test environment is out of the question for all kinds of legal reasons. Meanwhile, pressure is being put on the test manager to find a solution for the problem of the overrun.

With a test project, it is important to pay special attention to the setup and maintenance of the infrastructure. In order to keep the focus on this during the test, there is a separate phase within the TMap life cycle model. It is a phase that runs parallel with the phases of Preparation, Specification, Execution and Completion. For some activities, there are dependencies between these and activities in the other TMap phases. This is explained later in this section in connection with the relevant activity itself.

Test environment

A suitable test environment is required for the testing of a test object.

Definition

A test environment is a composition of parts, such as hardware and software, connections, environment data, maintenance tools and management processes in which a test is carried out.

Hardware refers to all the tangible parts of a computer (screen, hard disk, network card, et cetera). Test environment software refers to all the programs that should be present on the available hardware in order to run the software under test, such as operating programs, DBMS, network and other support programs. Connections are everything that is required to allow the test object to communicate with other systems. The environment data is the set of data that the test environment requires to be able to work with these (user profiles, network addresses, root tables, et cetera). Maintenance tools are tools that are required specifically to keep the test environment operational, and management processes are all the activities that are carried out around the setup and maintenance of a test environment.

Testtools

Definition

A test tool is an automated instrument that supports one or more test activities, such as planning, control, specification and execution.

Test tools can be used as instruments for achieving higher productivity and/or effectivity on the part of the testers and the testing. With the use of test tools, the emphasis is on "support" (see the definition). This means that a test tool is only a tool if the use of it delivers something; using a tool should not be a goal in itself.

One of the conditions for the successful use of test tools is the presence of a structured test method of operation. In a well-managed process, tools can certainly deliver significant added value, but they are counterproductive in an inadequately managed test process. The reason for this is that automation (what test tools actually do) requires a certain repeatability and standardization of the activities to be supported. An unstructured process cannot meet these conditions. However, the deployment of test tools can function as leverage for implementing a structured approach. Structuring and automation should therefore go hand in hand, in short: "Structure and Tool".

Workplaces

One of the aspects that are often forgotten in testing is the provision of a workplace, where testers can perform their tasks effectively and efficiently in satisfactory conditions. This involves an office setup in the most general sense, and so the workplace consists of more than just office space and a PC. Issues, too, such as e.g. entry passes, power supply and lunch-break facilities all have to be arranged.

If the testing is carried out in the framework of a project, extra office space should be organized. It is advisable to bring the test team together in one location (a room or a floor). This will form a basis for good mutual co-operation and coordination within the team. If that is not possible, the location of team members in various rooms should correspond with, for example, the allocation of the various system parts to the testers, the test types to be applied, et cetera. If developer and tester work together in multidisciplinary teams, they should be situated together in one location.

As with every project activity, a great deal of consultation takes place in testing. Because testing finds itself at the crossroads of the various activities in the project, testers have a lot of contact with the various groups (such as designers, programmers, administrators and users). It is advisable to place the test team in the vicinity of these groups. There are examples of improved test processes thanks to the relocation of the test team to the physical 'middle' of the project organization. This resulted in, among other things, increased mutual respect between the testers and other project participants, which benefited the quality.

The workplace intended for a tester at first sight does not differ much from the standard workplace. But appearances are deceiving. What is being tested is often new to the organization and the workplace. Testers may find themselves in a situation in which their workplace is unprepared for the new software. It is therefore often necessary to arrange separate authorizations for testers. For example, testers should have the possibility of installing the new software on their local PC, and this may also be necessary in order to use particular test tools.

Tip

Certain test varieties can deliver a great deal of data. An example of this is the performance test, in which a test tool is used. The output of this test tool may consist of thousands of lines of information. Stored in files, this may well grow to several gigabytes per test, often with printouts of over a hundred pages. It is therefore a good idea to adopt separate measures for dealing with this. For example, extra disk space could be reserved and an extra printer connected.

Preconditions

Before the setup and maintenance of infrastructure phase can be started, the description of the required infrastructure at an overall level, including the general plan, should be known and established in the test plan and/or master test plan. If test tools are being used, it should be known how the various activities within TMap are to be performed.

Method of operation

On the basis of the definition of the infrastructure set out in the test plan, it is considered whether closer specification and more detail are necessary. Besides the description of the required resources, it is also described what is expected of the suppliers during the maintenance of these resources. Since different expertise is required, the realization of the infrastructure is often carried out by other parties. From within the test project, the progress of the realization is monitored and if the progress is threatened, actions are devised. The realization should be completed before the Execution phase begins, but preferably earlier. Simultaneously with the realization of the infrastructure, a checklist is created that includes specific checks. This is used to determine, upon delivery, whether the infrastructure supplied meets the previously set requirements. After delivery, the infrastructure should be kept available for the testers at the quality level determined at the start of the phase. At the end of the test assignment, it is examined which parts of the infrastructure should be preserved. These can then be reused in future (re)tests.

Roles / responsibilities

It is advisable to delegate the organization of this phase to someone other than the test manager. This individual then takes the role of test infrastructure coordinator.

Activities

The basis of the Setting up and maintaining infrastructure phase is defined in the Planning phase. Here, within the activity "Defining the infrastructure" the infrastructure required at overall level is described, including the planning. This description (from the master test plan or test plan), serves as input for the first activity in this phase.

The Setting up and maintaining infrastructure phase consists of the following six activities:

1. Specifying the infrastructure
2. Realizing the infrastructure
3. Specifying the infrastructure intake
4. Intake of the infrastructure
5. Maintaining the infrastructure
6. Preserving the infrastructure

Figure 51 ("Setting up and maintaining infrastructure") indicates the sequence and dependency between the various activities. Activities "Realizing the infrastructure" and "Specifying the infrastructure intake" can be carried out in parallel. The dependency between the end of activity "Intake of the infrastructure" and the start of the Execution phase is significant. Before the test execution can start, there must be a correctly operating test infrastructure.

That is why it is essential to plan activity "Intake of the infrastructure" before the start of the test execution. It is even advisable to plan this well in advance (and the preceding activities as well) in order to prevent any start up problems with the test infrastructure from causing the test execution to overrun. Test execution often finds itself on the critical path of the entire project, and so problems with the test infrastructure indirectly cause the project

to overrun. Also, an operational infrastructure is very handy in the Specification phase. Test scripts can be tried out, and test data (in files, for example) implemented.

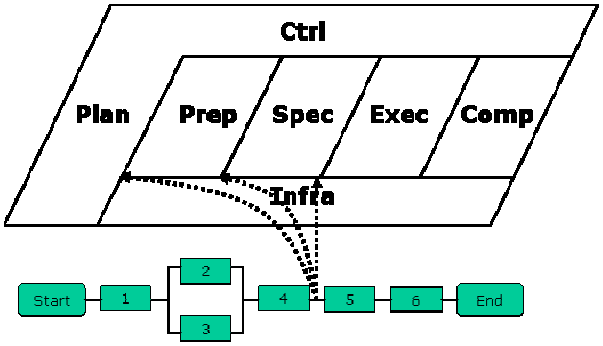


Figure 51. Setting up and maintaining infrastructure

In the definition of test infrastructure, it says that a distinction is made between test environment, test tool and workplace. For each of these three, the activities model, as previously described, should be followed. The activities of the three parts have a mutual relationship as regards timeliness. Activity "Intake of the infrastructure" plays an important role here. The intake of the infrastructure forms the link with the other phases in TMap and is also the common link between the three parts.

It is advisable to organize the workplace as quickly as possible, and it should be ready before the testers arrive. This means it should be prepared during the Planning phase. Often it is even necessary to have the workplace operational before the intake of the test environment can begin. And, in turn, the test environment often needs to be operational before a start can be made on the intake of the test tool. This is made clear in figure 51 "Setting up and maintaining infrastructure". The setup and maintenance of the infrastructure is a very complex operation, with many internal and external dependencies. The organizing demands close attention and it is therefore advisable to arrange this with the test infrastructure coordinator.

Tip

When test tools are used for automating the test execution the operational infrastructure must be in place before the Specification phase starts. This means activity "Intake of the infrastructure" is completed before the Specification phase starts. This is because in the Specification phase the automated test scripts are programmed and therefore you need an operational workplace, an operational test tool and an operational test environment.

4.3.4.1 Specifying the infrastructure

Aim

To specify the description of the required infrastructure (from the master test plan or test plan) in a more detailed level.

General method of operation

On the basis of the specification of the infrastructure contained in the (master) test plan, it is considered whether further specification and detail are necessary. The planning of the test environment, test tools and workplace is also worked out in more detail. Besides describing what resources are necessary, expectations are also set out in respect of the

supplying party during the management of these resources. The timely involvement of the various parties is essential. Agreements should be made for the supply and build of the infrastructure, and these agreements should be checked at regular intervals. In consultation with the various suppliers (internal and external) it is determined how detailed the specification should be. The delivery times of the various parts are included in the detailed plan.

Workplace method of operation

The specification of the workplace covers tangible subjects, such as required locations, desks, chairs, telephones, PCs, et cetera. But it also covers less tangible things, such as required authorizations, disk space, software, e-mail accounts, et cetera. The realization of these aspects may take a considerable amount of time. Occasionally it requires a special setup (e.g. project rooms) or special installation (e.g. the PCs). In other cases, items have to be ordered. It is advisable to emphasize at this stage specific requirements that are set in respect of management of the workplace. For example, obtaining separate status for the testers in the solving of problems in the workplace. This can be useful, since testers are no 'ordinary users' and sometimes require a different kind of support.

Test environment method of operation

In specifying the test environment, the various elements of the test environment should be considered. Definitions can vary among suppliers and organizations. For that reason, it should always be discussed clearly what is meant by particular terms. Another important point is the number of test environments required and the various types there are. Each type of test environment has its own purpose, with specific requirements applying to it.

The specifying of the technical form of the test environment should be done in consultation with someone who has technical knowledge of the environments. This individual should translate the concrete requirements (based on the aim of the test served by the test environment) into the technical form. As a basis for this, an architectural overview can be created, for example. This can be a difficult process, since two worlds (testing and technology) speak two different languages. It is up to the test-team individual responsible (the test manager or test infrastructure coordinator) to check whether this is organized satisfactorily.

Besides requirements concerning the setup of the test environment, requirements should be set in respect of the maintenance of it. Examples of requirements are:

- The backup activities that have to be carried out
- The comprehensibility of the software versions present
- The interfaces present
- The ability to change the test environment
- The ability to change the system date
- The use and management of test data
- Authorizations and their administration
- The required timetable for the building of a test environment.

Agreements should also be made at this point on how the test environment will be tested (see also the activity intake of the infrastructure). Other agreements may concern the contact with suppliers (direct by the test team or via another party) and how to deal with licenses.

Example

For the testing of a new customer administration, the following requirements were set during the specification in respect of the test environment and the maintenance of it:

- Backups are made upon request of the testers and take no longer than 15 minutes
- No changes are implemented in the environment without the explicit permission by mail from the test coordinator
- Created backups are returned upon the test coordinator's request within 15 minutes
- The resetting and securing of environments takes place between 20:00 and 06:00
- The operating system for the test environment is the same as that of the production environment
- Connection of system X and Y to the test environment should be available between 06:00 and 20:00
- Connection of system Z to the test environment should be available between 06:00 and 20:00 within 15 minutes of the request
- Connection with system W is simulated by a stub
- Testers have direct access to tables in the database (reading permissions)
- The system date should be open to change by the test team
- It should be possible to store 4 versions of test files
- Tool A should be available for the creating or copying of complete test cases.

Tip

In some organizations, a standard set of test environments is used and the test manager has to use these for his test. If that is the case, during this activity he investigates the specific characteristics of these test environments and how they fit within the test programme.

Test tool method of operation

If, in the creation of the (master) test plan, it is decided to employ test tools, this should be firmed up during this activity. The decision should be backed up by definite choices of one or more tools. As made clear in the definition of test tools, they are intended to support one or more test activities. During this specification of the test tools, it should be clear which test activities are to be supported and how this should be done.

Products

Detailed specification of workplace
Detailed specification of test environment
Test tool(s) plan of approach

Techniques

Not applicable.

Tools

Not applicable.

4.3.4.2 Realizing of the infrastructure

Aim

To realize the infrastructure according to the detailed specification from the previous activity.

Method of operation

The infrastructure is realized during this activity. The required hardware and software are purchased or ordered as necessary. The workplaces and test environment are organized and the test tools installed and configured. During this activity the framework of the test suite is built when tools are used for the automation of the test execution. Since all of these activities require special expertise, it is usually carried out by parties other than the testers. From within the test project (the test infrastructure coordinator) the progress of the realization should be monitored, in case it is threatened.

This activity should be carried out in parallel with the first phases of TMap and should be ready at the latest by the end of the Specification phase (preferably before, since time is required for the next activity, "Intake of the infrastructure"). When the activity is carried out depends on the part that is being realized and on the dependencies between the various parts. For example, the workplace should be realized first, preferably in the Planning phase. The realization of the test environment often takes a lot of time and therefore should be started quickly. But the situation can arise in which a workplace is necessary for the realization of the test environment. In that case, it is necessary to wait until the workplace is ready. If the test tool uses the test environment, the installation and configuration can only start when the test environment is ready. Otherwise, it is best to start this as quickly as possible. When tools are used for executing the test the realization of the infrastructure must be finished before the Specification phase starts. In the Specification phase the tools are used for creating the automated test scripts.

Both internal and external parties (e.g. the supplier of the test tool) play a part here. This makes it a difficult activity to manage, demanding good coordination. The infrastructure coordinator should check the progress and quality of the work supplied. The following sub activities should be carried out, for example:

- Check whether all the agreements are still valid
- Have bottlenecks and problems solved and adopted measures established in new agreements
- Check installations. The created checklists can be used for this (where possible) for purposes of the infrastructure intake.

Products

Operational workplace
Operational test environment
Installed test tools

Techniques

Not applicable.

Tools

Not applicable.

4.3.4.3 Specifying the infrastructure intake

Aim

To specify the method whereby the intake of the infrastructure is carried out.

Method of operation

Because the infrastructure is often supplied by parties other than the test team and because it plays a very important role within the rest of the testing, it is important to designate a formal acceptance point. At this point, it will be determined whether the products will serve the intended purposes and whether they meet the previously set requirements. (It is a kind of acceptance test of the infrastructure.) This takes place by means of an intake: an activity in which, on the basis of a checklist, it is determined whether the workplace, the test environment and the test tool are functioning and whether they meet the previously set requirements.

The checklist is drawn up on the basis of the specifications of the various parts. It should be available before the end of the previous activity (realizing the infrastructure), but preferably earlier, so that it can be used during the realization for interim checks.

This activity bears a close relation to the activity of "Specification of the test object intake" in the Specification phase. There are situations in which certain aspects of the infrastructure can only be checked with the aid of the test object or an early or interim version thereof. For example, a release procedure for the test environment can only be checked with the test object. But the correct installation of a test tool for the automation of the execution, too, can only be checked with the test object.

Example

The following checks can be carried out for the workplace:
- Are the required PCs, printers, workplaces, telephone lines, routers, et cetera present and correctly installed?
- Is the required system software installed?
- Is the system software the right version?

The following can be carried out for the test environment:
- Has access to the test environment been provided?
- Has access to the application been provided?
- Has access to the database been provided?
- Has the database been filled with the correct data (e.g. a copy of production)?
- Have all the authorizations been provided?

The following checks can be carried out for the test tools:
- Are all the licenses operational?
- Can the test tool be accessed from every workplace?
- Is the connection between test tool and test object operational?

Products

Checklist "Workplace intake"
Checklist "Test environment intake"
Checklist "Test tools intake"
Intake procedure

Techniques

Not applicable.

Tools

Testware management tool.

4.3.4.4 Intake of the infrastructure

Aim

To carry out the intake as prepared in the preceding activity.

Method of operation

All the checks on the checklist, created during the preceding activity, are gone through. This determines whether the test environment, test tool and workplace function and whether they meet the previously set requirements. Any missing parts are reported to the stakeholders by means of a defects report. These parts should of course then be made available as quickly as possible. Missing parts in the test environment will have a delaying effect and have an impact on the entire project. The Execution phase is often on the critical path, and if it cannot start (because for example the test environment is not functioning), the entire project will be delayed. The intake should not be underestimated, and should be carried out as quickly as possible. The intake of the test environment is preferably carried out during the Specification phase. If this is not possible, then it should be done at the start of the Execution phase, at the latest. This may be the case if the test object is required and is only available at that time.

Products

Defects
Operational and usable workplace
Operational and usable test environment
Operational and usable test tool
Intake report

Techniques

Not applicable.

Tools

Testware management tool
Defect management tool.

4.3.4.5 Maintaining the infrastructure

Aim

To keep the infrastructure (test environment, test tools and workplaces) available for the testers at a consistent level of quality.

Workplace method of operation

Maintaining the workplace, so that it is and remains available to the testers, is usually an activity that is organized as standard within other maintenance activities in an organization. As regards the PC in the workplace, it is important that the usual

maintenance organizations know that this is specially intended for the testers, since it can mean that other agreements apply concerning, for example, authorizations and prioritization in problem solving.

Test tool method of operation

The test tool can be maintained within the test project by the testers who use it, but also by a separate maintenance department (e.g. a permanent test organization). An important maintenance element is the regular checking for new versions of the test tool and then providing this to the users. Besides this, the management activities apply as described for the test environment and for the test tool.

Test environment method of operation

The supply of the test environment on an ongoing basis, so that the testers are able to carry out their test cases and analyze their findings, covers a range of activities. These take place during the Execution phase. Examples of these are:

- Solving bottlenecks
- Provision for logging
- Backup and restore
- Implementing changes
- Monitoring.

Solving bottlenecks

The execution of test scripts may be delayed if problems occur in the test environment (e.g.: a batch program has not run). Since the execution of test scripts is often on the critical path of a project, it is important to give the highest priority to solving these bottlenecks.

Example

At a government institution, a project has a fixed deadline because the solution is related to a change in legislation that is to be implemented by a certain date. The Execution phase is on the critical path of this project and it is therefore in everyone's interest that this phase is not delayed. In consultation with the maintenance department, it is therefore agreed that the infrastructure that the testers use will be given a so-called "production" status. This means that the management department deals with bottlenecks experienced by the testers with the same priority as if it concerned the production environment. This is justified by the fact that, should the project not be completed on time, the legislation could not be implemented and so the primary process could no longer proceed. This separate status only applies during the test execution.

Provision for logging

Systems can provide information in the form of logging, which can be used in retrospect to check the actions that have been carried out. The logging is an important source of information for testers in the analysis of their findings. The provision of this information is therefore also an important activity. It may be decided to make it available on request, but another (less labor-intensive) variant is to give the testers themselves access to the logging.

Backup and restore

Particularly in respect of infrastructure used by testers, it is important to secure the data by means of regular backups. This may be for the purpose of securing starting situations and using them repeatedly for the test, but also of investigating particular defects. This

concerns not only backups of the test environment, but also of test tools and the PCs in the workplace.

Tip

Always test the backup and restore procedure before the test starts. The way to do this is to is to restore the backup immediately the first time it is created. This way the backup procedure and restore procedure are tested.

Implementing changes

During the project, the test environment is subject to changes owing to all kinds of internal and external causes, for example:

- Phased delivery and changes in the test environment
- Delivery or redelivery of (parts of) the test object
- New or changed procedures
- Changes in the simulation and system software
- Changes in the equipment, protocols, parameters, et cetera
- New or changed test tools
- Changes in the test files, tables, etc.:
  - Conversion of test input files to a new format
  - Reorganization of test files
  - Changes in nomenclature

Changes in the test environment should only be implemented following permission from the test management. Depending on the nature and size of the change, this will be made known generally to the test team. A new intake will then take place in the test environment.

Tip

A pitfall in the planning is to assume that the installation of a new version of the test object takes no time. In a particular project, the first couple of versions took weeks because of the great complexity and instability of the entire test environment and test object. Later, this was optimized and subsequently never took more than a few days each time.

Monitoring

The situation can occasionally arise in which a defect requires further research and deeper technical knowledge than the tester has at his disposal. Assistance can be called upon and he can 'help to look' at a technical level (monitoring) at what happens in conjunction with certain actions.

Products

Operational and maintained test infrastructure
Defects test infrastructure

Techniques

Not applicable.

Tools

Not applicable.

4.3.4.6 Preserving the infrastructure

Aim

The aim of this activity is the identification, updating and transferring of the infrastructure under maintenance, in such a way that it can be used again in future (re)tests. This activity is optional.

Method of operation

This activity starts simultaneously with the Completion phase and covers the following subactivities:

- Selecting the infrastructure
- Collecting and refining the infrastructure
- Transferring the infrastructure.

Selecting the infrastructure

In consultation with the future maintenance department of the infrastructure, an inventory is drawn up of which parts are now actually used (the configuration) and what is 'worth' transferring. The decision should be made based on the consideration of what it costs to keep and maintain the infrastructure, and what it would cost to realize the infrastructure again at a later stage. Besides this, there is the possibility that certain software or hardware (such as parts of the test environment, but also certain test tools) are only of use during the initial phase of the testing and are no longer necessary. It is then a waste of effort taking this under maintenance. This identification can also clarify the difference between the specified infrastructure and the infrastructure actually used. There can be discrepancies here (certain software or hardware that was set up but never used) and this point of learning can be taken forward into the evaluation of the test process.

Collecting and refining the infrastructure

The description of the infrastructure in the "Detailed specification of the infrastructure" should be adapted to the configuration that is to be transferred. This is of essential importance, as otherwise everything will have to be created anew for future tests. It is important with this description to look carefully at the configuration of the workplaces. In this "Detailed specification of the infrastructure" a list is included containing the components that are transferred. Components may be licenses, environment data, scripts, software, tools, registry files, hardware, accounts, databases, files, etc.

Transferring the infrastructure

Finally, the actual transfer of the infrastructure takes place. The configuration is transferred according to the adapted list in the document "Detailed specification of the infrastructure".

Products

Preserved test infrastructure

Techniques

Not applicable.

Tools

Not applicable.

4.3.5 Preparation phase

Aim

To obtain, with the client's agreement, a test basis that is of sufficient quality for designing the test cases. In order to determine this, a testability review of the test basis is carried out during this phase, which will provide insight into the testability of the system.

Definition Testability is the ease and speed with which characteristics of the system can be tested (following each adjustment).

Early defect detection

There is another reason for assessing or evaluating the test basis, apart from establishing its testability. Evaluation activities can reveal potentially expensive defects at an early stage of the development and test processes. The test basis forms the blueprint for the new system to be built. Anything that is not mentioned in the test basis is left to the development team to solve. The development team goes to work on developing the new information system on the basis of the system documentation, which may contain mistakes. If these are not found in time, it can lead to a lot of (often expensive) corrective work. The sooner a mistake is found in a development process, the simpler (and cheaper) it can be reworked [Boehm, 1981]. If, for example, a defect in a specification or requirement is not discovered until the execution of the acceptance test, the reworking costs are high. Not only must the software be amended, but also, for example, the technical and functional designs. In general, it appears that early defect detection makes savings of 50%-80% possible.
By assessing the test basis and detecting defects early, the quality of the test basis will increase.

Practical example

In the real-world examples below the testability review was carried out as an activity of evaluation:

- A supplier of packages has achieved a return-on-investment of 10:1 through early testing of the designs. Because of this, €21.4 million is saved annually on project costs, and the average time-to-market has been reduced by 1.8 months.
- A company in the telecommunications sector avoids 33 hours of reworking per defect by evaluating the code.
- A large computer manufacturer saves 20 hours of test effort and 82 hours of reworking for every hour spent on inspections.
- A multinational in the chemical sector spends 10 times less maintenance money on 400 inspected software products than on 400 non-inspected software products.

Context

While both the (definition of the) test basis and the agreed test strategy are specified in the test plan, the test basis is often not yet available at the time of creating the test plan. In the Preparation phase, it has to be investigated whether the test basis delivered corresponds with, and is usable for, the previously established agreements in the plan. If this does not appear to be the case, it may be necessary to adjust the plan, which can have both a negative and a positive influence on one or all of the money, time and quality aspects.

Negative influences are, for example:
- the lack of a definitive test basis
- a qualitatively inadequate test basis
- a test basis with more complex algorithms than expected.

Positive influences are, for example:
- a test basis with less complex algorithms than expected
- a test basis that anticipates the making of logical test cases (see tip).

Amending the plan is an activity from the "Control phase" and is further explained in that section.

Tip

A government organization decided to have the designers supplement the functional design with decision tables. The idea behind this was that the designers themselves knew the intention of the design better than the testers, who had to create the (logical) test cases based on the design. Since the testers were thus given a 'head start' and needed to investigate less, the organization reduced the amount of time by 25% in the Specification phase.

Preconditions

The Preparation phase starts as early as possible following the consolidation of the test plan *and* after the consolidated test basis is made available (see "In more detail").

In more detail

The test basis is consolidated when the client indicates that enough activities have been carried out that guarantee the quality of the specifications and other information. Consolidation of the specifications is of great importance, since they form the basis for both the testers and the developers and may subsequently only be changed by means of formal change procedures.
While, in principle, only the client may consolidate the test basis, situations are conceivable in which the test manager considers proceeding as though a test basis has been consolidated. For example, because the test manager doesn't want to hinder the progress of the test, or if testers are in danger of 'being freed up'. In making such a decision, it is important to make clear agreements on this with the client. There is a good chance that the test basis will change, with the possible consequence that previously created test designs have to be amended. This can lead to extra costs and extension of the timeline. It should be established in the agreements with the client how this is to be dealt with, so that there is no need for discussion in retrospect.

Method of operation

Once the test basis has been put at the disposal of the test team, a start is made on its testability review. It is first examined whether the summarized information, of which the test basis consists, is still correct. If necessary, it is brought up to date in consultation with the client. During this examination, it may appear that all of the information is not yet available for the tester, or perhaps will not be arriving at all. In such a situation, a way must be found of obtaining the missing information.

When the test basis is clear, this is assessed from the testing perspective for e.g. consistency, understandability and completeness. Subsequently, on the basis of checklists, an assessment is made as to the extent to which the established test strategy and

associated test (design) techniques are applicable. The conclusions are documented in a testability review report and discussed with the client. The results of this report may give rise to adjustments to the test basis, the test strategy and the test techniques to be employed.

Tip

Synergy between evaluation and development / testing process
In some organizations, design specifications are structurally evaluated before a subsequent development phase is started. By making the various points of focus from the Preparation phase part of such an evaluation, a satisfactory degree of synergy is created between the structural evaluation and the test activities from the Preparation phase. In this situation, one or more members of the test team participate in the evaluation process. They take responsibility for the aspect of testability in relation to the design specifications. The testers can also take the initiative of introducing a structural evaluation process (requirements being, for example, set out in a SMART[6] framework), using evaluation techniques as described in section 4.12 "Evaluation techniques". Evaluation then becomes an integral part of the method of operation. In the execution of the evaluation activities, use can of course be made of the various checklists as described in the activity "Creating checklists".

Roles / responsibilities

The testability review report is created by the test manager or test coordinator. All the other activities can be carried out by any of the test team members. The report is intended for the commissioner of the test (the client).

Activities

The Preparation phase consists of the following activities:
1. Collection of the test basis
2. Creating checklists
3. Assessing the test basis
4. Creating the testability review report.

The diagram below depicts the sequence and dependencies between the various activities:

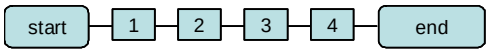


Figure 52. Preparation phase

[6] SMART: S=Specific, M=Measurable, A=Achievable, R=Realistic, T=Time-bound

4.3.5.1 Collection of the test basis

Aim

The collection of, the definitive, if necessary overhauled, test basis is established in consultation with the client.

Method of operation

The definition of the relevant information for the execution of the test is in principle already established in the test plan (e.g. functional and technical designs, requirements, use cases, user manuals, interview reports, prototype, and reference system). However, it is possible that, in respect of the exit information, changes have taken place. The test plan should then be amended and the identification of the information reviewed. Finally, the various parts of the test basis are actually collected. Eventually, of course, the test team should have the correct (version of the) test basis at its disposal.

A point to bear in mind here is that the test basis does not always have to be present, complete, up to date, or established in documentation. A test basis often appears to be incomplete because, for example, non-functional requirements have not been specified, while they are nevertheless considered to be risk-related. By alerting the project to this, a (timely) trigger is created for bringing it to attention.

Alternative test basis

If test-basis problems do indeed arise, some solutions obtained from practice for obtaining an alternative test basis are listed below:

- Present system in production as reference system
  Supposing the system documentation is missing, obsolete or incomplete in a conversion or migration project, for example. The creating, supplementing or updating of this documentation normally does not belong within the scope of the project. In such a situation, the present production version of the system is used as test basis. This is a particularly good alternative in situations that involve few or no changes to the functional operation of the system, or if the changes are well documented.
- Prototype as test basis
  In a situation that does not accord high priority to the production of system documents, which are possibly only to be delivered at the end of the project, a prototype is sometimes made. This occurs, for example, with Rapid Application Development or variant of this (including SP, DSDM and RUP). Since the prototype is often made in co-operation with the user, this can also be used as the test basis.
- Information session
  During, for example, maintenance operations, it often appears that neither the system in production nor the changes to it have been well documented. The organization of information sessions for everyone involved (developers, designers, users, administrators, etc.) is a good way of clarifying both the operation of particular system parts and the changes to be implemented. The information obtained during such a session can be used as a test basis.
- System documentation from the last-but-one iteration as a test basis
  With iterative and incremental system development approaches, there is a possibility that the system documentation will only become available to the tester at a later stage. In a situation where it is not permissible to change the system documentation during the last iteration, the test basis is made available to the tester at the end of the last-but-one iteration. In the situation where it is permissible to change the system documentation during the last iteration, it may be considered whether to use the system documentation from the last-but-one iteration as the test basis (often more than

80% ready). At the end of the last iteration, the – often small – changes to the system documentation have to be processed in the test cases by the tester.

An important point in connection with the above means of obtaining an alternative test basis is that this is seen by the client (and any other stakeholders) as the test basis. However, a test basis obtained in this manner will seldom be approved or consolidated. It is therefore important for the client and the tester to be aware of the risks that this involves. It is advisable not only to inventory these risks, but also to establish the associated countermeasures. For example, who has the 'deciding vote' if it appears that the realized functionality of a (sub)system differs from expectations based on the alternative test basis?

Occasionally, so little information is present that even establishing an alternative test basis is impossible. In such a situation, other sources of information may be resorted to, and while they cannot be used as an alternative test basis, they are perfectly usable for, for example, deriving logical test cases (see tips on "Absence of test basis").

Tips

Absence of test basis
If no test basis is present, the tester should go in search of other sources of information that can serve as a basis for creating test cases. Bach, Whittaker and Kaner have devised an approach for this:

- HICCUPP [Bach, 2003]
  Information for creating test cases may be obtained, for example, from norms and standards, memos, user manuals, interviews, advertisements or rival products. Bach has set this out in his HICCUPP approach:
  History. Is the present operation of the software consistent with the previous operation?
  Image. Is the operation of the software consistent with the image of the organization?
  Comparable. Is the operation of the software consistent with that of other comparable products?
  Claims. Is the operation of the software consistent with how people say it should operate?
  User expectations. Is the operation of the software consistent with what we (the testers) think the user wants?
  Product. Is the operation of specific software components consistent with comparable software components within the product?
  Purpose. Is the operation of the software consistent with the apparent aim of the software?

- 18 Attacks, by Whittaker and Jorgenson [Whittaker, 2000]
  Some software defects are so trivial that good standard tests (attacks) can be defined for them. The 18 attacks of Whittaker and Jorgenson listed below can form an excellent basis for creating tests or be used to supplement existing tests:
  User interface (input)
  1. Generate input that will provoke all the error messages.
  2. Generate input that will require all the default values to be entered.
  3. Try to enter all the permitted symbols and data types.
  4. Enter too many symbols.
  5. Find correlations between input fields and test combinations of their values.
  6. Enter the same data repeatedly.

  User interface (output)
  7. Try every possible output for every input.
  8. Try to cause incorrect output.
  9. Try to change characteristics/values of the output.

10. Refresh the screen.

Stored data

11. Enter data from every possible starting point.
12. Try to save too many or too few characters in the database.
13. Try to find alternative ways of changing internal data restrictions.

Calculations

14. Try out incorrect operand and operator combinations.
15. Try to get a calculation module to invoke itself.
16. Try to make the resulting values too high or too low.
17. Try to find functions that make use of the same data.

System interface (media)

18a. Make all the storage space unavailable.
18b. Make the system busy or unavailable.
18c. Damage the system.

System interface (files)

18d. Allocate an incorrect file name.
18e. Change the permissions (including reading and writing permissions) of a file.
18f. Change the content of a file, or corrupt it.

- Kaner's 480 bugs [Kaner, 1999]
  Kaner has created a list of common software defects. This list can be used to find the same or similar defects in the software under test. Alternatively, the list can be used in a more general sense for:

  Gathering test ideas

  Investigate whether a defect on the list could arise in the software under test. If this is theoretically possible, consider how you might find it. Then create test cases (or not) depending on the damage the defect could cause in production.

  Test design review

  Select a few test situations from the test design and find a possible defect from the list for each test situation. Then examine, for each possible defect, whether it could occur in the software under test and whether it would then be found by the test cases created.

  Wider perspective

  Check the list for types of defects that are often overlooked (out-of-the-box thinking).

  Training

  Show new testers what can go wrong and have them create test cases with which these defects can be found.

When using one or more of the approaches mentioned with a view to arriving at an alternative test basis, or a basis for deriving test cases, the tester would do well to bear in mind that it is not the tester's job to create the test basis. The tester assesses and uses the test basis exclusively for testing purposes. The creation of system documents was, is and remains the responsibility of e.g. the project or the development department. The tester should avoid sitting in the place of the designer. This means that the test basis that is obtained from one of the above-mentioned approaches should always be agreed with all the stakeholders, on the one hand to confirm the way the system should function and/or be built, and on the other hand to confirm agreement that this is indeed the alternative test basis against which testing is to be carried out, or the basis from which test cases should be derived.

Products

Consolidated test basis.

Techniques

HICCUPP [Bach, 2003].
18 Attacks by Whittaker and Jorgenson [Whittaker, 2000].
Kaner's 480 Bugs [Kaner, 1999].

Tools

Not applicable.

4.3.5.2 Creating checklists

Aim

The checklists are created, on the basis of the test strategy laid down in the test plan, for the various part objects/characteristics under test. These checklists form a guide in assessing the test basis.

Method of operation

With the aid of checklists, the test basis is checked for testability. During this activity, the checklists needed for the testing are created. Depending on the selected test design techniques, test types, information sources that determine the test basis and the part objects/characteristics under test, one or more checklists should be created (see also the tip "Test design techniques in the absence of a test basis" below). Each checklist should indicate which specific verification aspects play a role in the testability review. If you wish to avoid duplicating work on identical parts of the test basis during the evaluation, the separate checklists could be consolidated into one checklist. In creating the checklist, use could be made of the general checklist of "test design techniques", to be found at www.tmap.net.

Partly owing to the diversity of test design techniques and information sources that determine the test basis, it is not possible to create one general checklist per part object/characteristic. Therefore, checklists should be created specific to the situation per organization and per project. It is advisable always to create a checklist, as in practice it often appears that too much attention is paid to the use of standards and correct spelling, or even to these aspects alone. This can be a cause of friction among the various people involved.

Tip

Test design techniques in the absence of a test basis
The test plan contains, among other things, a summary of the information of which the test basis consists, as well as the test strategy. However, if it appears that the agreed (documented) test basis is partly or entirely lacking, it may be that the testing has to be carried out on the basis of different (non-documented) information. In that case, not all the test design techniques are suitable.
Some coverage types and test design techniques that are often suitable in such a situation are:

- Data combination test
- Error guessing
- Exploratory testing
- Boundary value analysis
- Checklist based.

For notes on these coverage types and test design techniques and the use of them, refer to chapter 3 "Website".

Products

Various checklists or one consolidated checklist for assessing the test basis.

Techniques

Checklist "Test design techniques" (www.tmap.net).

Tools

Not applicable.

4.3.5.3 Assessing the test basis

Aim

To establish the testability of the test basis. Testability here means completeness, consistency, accessibility and translatability into test cases.

Method of operation

The test basis is assessed using evaluation techniques and the previously created checklist(s) to obtain insight into the applicability of the established test strategy and related test design techniques. If it appears that the test basis falls short, it is of course important to report this to the supplier of the test basis via the client as quickly as possible. This party can then take responsibility for clarifying and/or filling in the gaps. The registration and flagging of these defects in the test basis take place by means of the procedures established in the activity "Organizing the management".

Products

Test basis defects.

Techniques

Checklist for assessing the test basis (product from "Creating checklists").
Evaluation techniques.

Tools

Defect management tool.

4.3.5.4 Creating the testability review report

Aim

The testability review report:
- provides feedback on the quality of the test basis and its impact on the planned test programme
- discusses the weak spots in the system design timely
- obtains information on project risks.

Method of operation

A testability review report is created based on the individual test basis defects. This report supplies a general summary in respect of the quality, or testability, of the test basis. Any consequences of inadequate quality should also be described. Discrepancies in respect of the summarized information in the test plan of which the test basis consists and the agreed test strategy are also described. This can give rise to adjustment to the plan in connection with, for example, the strategy to be followed and the test techniques to be employed. For further explanation of this, refer to the "Control phase".

The testability review report could consist of, for example, the following sections:
- Formulation of the assignment
  An identification of the original (or, if necessary, amended) test basis and a description of the client and the contractor.
- Conclusion
  The final conclusion in respect of the testability of the examined test basis and any related consequences or risks: is the test basis of sufficient quality to justify starting on specifying tests as established in the (amended) test strategy?
- Recommendations
  Recommendations in respect of the assessed test basis and any structural recommendations with an eye to producing a better test basis in the future.
- Defects
  The defects found are described in detail or reference is made to the associated defects forms.
- Appendices
  The checklists used.

Products

Testability review report.

Techniques

Not applicable.

Tools

Not applicable.

4.3.6 Specification phase

Aim

During the Specification phase, the required tests and starting points are specified. The aim is to have as much as possible prepared, in order to be able to run the test as quickly as possible when the developers deliver the test object.

Context

This phase begins when the testability review has been carried out on the test basis and the defects in it have been processed as far as possible. The test specification runs in parallel with the completion of the software (or parameterization, in the case of packages). The software is the primary product of the development process and is usually also on the critical path of the process. The focus of the (project) management is therefore upon this. The test specification is only of indirect interest, but this changes at the point when the software is transferred for the test execution and the attention of the (project) management is then drawn to it. The test team has to be ready then to start the test execution. The test specification is aimed at preparing as much as possible so that the test execution can be performed as fast as possible and be on the critical path for as short a period as possible.
The test manager has to be aware of this. He should translate, as far as possible, the signals given by the test specification problems into consequences (in terms of time, finance and quality) for future test execution and the total productive process.

Preconditions

The following preconditions should be met before the Specification phase can be started:

- The test basis is available and placed under configuration management
- Defects from the testability review have been processed.

Method of operation

During the Specification phase, the testers specify the required tests per test unit. This is done by creating checklists or specifying test cases on the basis of the allocated test approaches, coverage types and/or test design techniques. When specifying test cases, the testers also create test scripts, in which the test cases are put into an efficiently executable sequence. On this basis, and partly in parallel with it, the testers define one or more central starting points for the testing that the test cases can use. This may be a copy of production or a central base table listing. A special form of a test to be specified is the test object intake. This test should check in the Execution phase whether the test object is sufficiently testable for a meaningful and efficient test execution.

Roles/responsibilities

The activities in the Specification phase are carried out by the testers.

Activities

Within the Specification phase, the following activities are distinguished:

1. Creating test specifications
2. Defining central starting point(s)
3. Specifying the test object intake.

The diagram below shows the sequence and the dependencies between the various activities. Activities 1 and 2 run in parallel, but mutually influence each other.

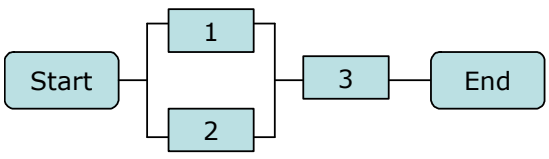


Figure 53. Specification Phase

4.3.6.1 Creating test specifications

Aim

The creation of the test specifications per test unit.

Method of operation

The testers specify the necessary tests for the test units in the test plan. After completion, the test specifications are placed under configuration management.

Definition A test unit is a collection of processes, transactions and/or functions that are tested collectively.

Depending on the test variety and test approach, coverage type and/or test technique selected for the test unit, this activity may consist of anything from the creation of a checklist to the design and specification of test cases according to a coverage type and/or test design technique or to the design of a test with other techniques. The possibilities are further explained below. Explanations are also given of a scalable regression test and of the relationship between this phase and exploratory testing.

In the course of this activity, problems may arise with the test basis. Roughly, these can be categorized as follows:

- Defects
  As with the testability review, the testers may find shortcomings and/or ambiguities in the test basis. The testers create a defect report on this. Via the defects procedure, it is passed to the test basis supplier, who can then solve it
- Absence of test basis
  If the testability review has been insufficiently executed, it may only appear at this stage that certain parts of the test basis are missing or not detailed enough, so that they are not, or not sufficiently, testable. The same types of measures as adopted with the testability review may be considered; see the section on "Preparation Phase".

  Tip

  With iterative or agile system development, the test basis is often not 100% complete at the start of the iteration, but is completed during the iteration. Besides the above-mentioned measures, it is advisable to carry out a minimal testability review with each addition to the test basis before specifying tests based on the addition.

- Unstable test basis
  If the supplier of the test basis makes regular changes to it, for example because of defects or change proposals, this makes for an unstable test basis. With every change, the testers have to examine the relevant test specifications to see whether adjustments are necessary. These reworking operations are always difficult to estimate in advance. The test manager is well advised to arrange a certain level of reserve budget and time for this when creating the test plan. If these are exceeded, the project management should be notified that more time and finances are required (see also section 6.3 "Control Phase"). Other possible measures are to defer the specifying of the tests for the unstable parts in the plan or to create the logical test cases, but to delay the physical makeup of them until the test basis is (more) stable.

Test design techniques

For the creation of the test cases test design techniques can be used. In Chapter 3 test design is explained in great detail.

Checklists

Besides the specifying of test cases, many tests take place with the aid of checklists. These are used with simple functional tests, but also for the evaluation of e.g. maintainability, manageability, user-friendliness or security. While a checklist is usually specific to the situation, testers often use a general checklist as a basis and make specific adjustments to this. The general checklists may be supplied from within the organization (by the test department) or from the literature or via the Internet. Various examples of checklists for testing certain quality characteristics can be found at www.tmap.net. The creation (and execution) of a checklist requires a competent tester with the necessary knowledge of the object part or characteristic under test. It is therefore advisable to have the checklist reviewed.

Other techniques

Apart from the specifying of test cases according to test design techniques and the creation of checklists, other techniques are possible that do not fall into either of the above categories. These techniques mainly apply to the testing of quality characteristics, such as portability, usability, performance and security.

Constructing and managing scalable regression tests

In more detail

In practice, regression tests are often inadequately set up. In this section, an approach is described for the constructing, using and managing of regression tests based on the *Test Cube* principle [Test Cube, 2006]. In this, connected principles are described, which make it possible to:

- Specify test cases and execute them based on priorities
- Report quickly and adequately on the progress of test specification and/or test execution
- Plan and estimate tests accurately
- Create fast and variable regression tests
- Process changes in the test object easily into the test.

The principle behind the Test Cube is that, per test case, a collection of supplementary data is established: the test cases within the test are 'classified'. With the aid of these classifications, selections can be made through all kinds of cross-sectioned subsets of test cases from the entire test.
Examples of classifications are:

- Application
- Object part
- Function
- Risk category
- Process (part)
- Release
- Requirement
- Transaction
- Test intensity.

The right selection of classifications and the correct classification of the test cases will determine the usability of this concept. Essential in this is the classification according to test

intensity. This classification indicates the 'weight' of the test case in the test and makes it possible to create a risk-based regression test of variable test intensity.
The application of these test intensity categories in the creation of regression tests is as follows (using three categories):

- By only selecting the test cases of an object part from category 1, a small regression test is created. This subset is used for an object part to which no amendments have been made (or for a pretest on a new or radically changed object part)
- The test cases from category 2 (includes category 1) deliver a normal regression test, for example for an object part in which amendments have been made
- The test cases from category 3 (includes categories 1 and 2) cover the entire object part and are applied to the new or radically changed object parts.

No requirements are set as regards the degree of detail in which the test cases are specified. If test cases are expected to be executed by testers who have no domain knowledge, the test cases should be written in more detail.

The concept only sets one specific requirement of its own in respect of the test cases, and that is that they should be independent of each other, as described for the creation of the physical test cases. This is the so-called independence principle of the concept. It should also be possible to execute the test cases in parallel with each other. Test cases that require exclusive use of the test environment for a specific period hamper the execution of other test cases. This in turn hampers the plans for the timeframe of the testing process.

Application of this concept facilitates measurement of the size of the (regression) test and associated activities in the test process.

As with the testware in general, careful consideration should be given here to when, how and by whom this test can be kept current.

For further explanation, refer to the relevant white paper [TestCube, 2006].

Session-based exploratory testing

In more detail

Exploratory testing (ET) is actually not purely a test design technique. With ET, the tester makes decisions during the test execution as to which test he is going to execute. He designs a test on the spot, using his knowledge of test design techniques, without documenting them. As such, ET has no place in the Specification phase, since everything takes place during test execution. The reason we are paying attention to it here is that, in order to make ET more manageable, it is often organized in the form of sessions with clear test goals that can be completed in a few hours. These test goals are known as test charters. While the list of test charters is dynamic, the testers are well advised to compile an initial list of test charters prior to the Execution phase.

Products

Test basis defects
Test specifications (checklists, test cases, test scripts).

Techniques

Approaches, coverage types and test design techniques (chapter 3)

Checklists for various quality characteristics, www.tmap.net

Tools

Defect management tool
Test design tool
Model-based testing tool
Testware management tool
Automated test execution tool
Performance, load and stress test tool.

4.3.6.2 Defining central starting point(s)

Aim

The defining of one or more central starting points from which the testers can obtain data for their test specifications.

Method of operation

A good starting point is of essential importance for the sake of being able to (re)test. This will contain everything necessary to prepare the test object and the test environment before starting with the test cases in the test script. This involves not only the test data required for the processing, but also the condition in which the system and its environment should be. It relates to, for example, the setting of a certain system date or the running of certain weekly and monthly batches that put the system into a particular condition.

In practice, incorrect starting points appear to be a significant source of problems for the testing. To avoid testing using the wrong starting points during the test execution, it should be considered at an early stage how these are to be constructed and which process is to be employed in using them. If this is not done, the following problems may arise:

- Non-reproducible test results
  If a test script is executed twice on the same version of the test object and the results vary, this may be the result of divergent test data in the starting point. Extra data may have been added to or removed from the starting point for other tests.
- Deteriorating starting point
  During the test execution, test data are used and amended. New data come into the system; existing data are amended or perhaps even removed. If no process exists to manage the starting point, nothing is known regarding its quality.
- Testing gets increasingly expensive
  If the starting point is of poor quality and is not documented anywhere, the testers are obliged to make increasing efforts (in seeking or creating test data) for the execution of the test cases. Moreover, the risk of mistakes on the part of the tester increases. This will increase further in time, as the starting point becomes increasingly less well known and therefore poorer.
- Insufficient information on defects causes delay
  The starting point takes an important place in the reporting of a defect. It clarifies a defect. If this starting point is not known during the analysis of the defect, delay will result. Developers themselves have to go in search of the original starting point or have to ask the tester for clarification.

In the test specifications, the necessary starting point is specified per test script. To avoid redundancy and to restrict the number of physical files needed, one or more central starting points are defined that the testers can use in the creation of their test cases.

The creation of central starting points can take place in parallel with the setting up of the test specifications and is often an iterative process. Often, a tester will start with a central starting point by, for example, proposing the contents of master files. Master files are data that drive the system, but are not part of the primary data processing. Examples are discount tables, tax percentages, postcode tables, product types and customer types. A subsequent step may be to propose an initial content of primary data, e.g. a number of customers, products, orders and invoices. It may be decided to define several central starting points, if this appears to be useful in specifying the tests. The difference may be the type of data, e.g. the one central starting point with all kinds of variations in customers, and the other with all kinds of variations in orders. Another possibility is a difference in time.

For example, a central starting point could be defined just before the year's end and just before disbursement of holiday pay, since these are significant testing points.

In addition, all kinds of starting points emerge in the creation of the test specifications, usually one per test script. The tester who manages the central starting point will consult on this with the tester of the starting point of the script as regards which data are suitable for adding to the central starting point. In this, the following criteria, for example, could be used:

- Can other testers reuse (part of) the starting point of the test script?
- Does the starting point of the script conflict with the (consistency of) the central starting point?
- Can including the starting point of the script in the central starting point disrupt other tests?
- Will including the starting point of the script in the central starting point lead to efficiency benefits in the execution of the script?

There are various possibilities for loading the central starting point with test data. These are described later in the book.

The description of the central starting points is created in accordance with the established norms and standards for testware and taken under configuration management after completion.

Naming test data

A point of focus when creating your own physical test data is the business of naming. It may be decided to name the data similar to those in production. In that case, realistic (although fictional) names are given to e.g. test customers, test addresses, test codes, test products, etc.

It may also be decided to give the data a name that is relevant to the test, for example by including the test-case number, test unit, object part or test goal in the name. This will also help with the solving of defects and transfer to other testers.

The third option is to generate meaningless names. For the foregoing example of test customers, then, these would be:

- Person1
- Person2
- Person3
- Person4
- Etc.

This last option saves time in searching for and creating realistic or test-related names, but also involves a risk. It may cause a certain functionality or other characteristic of the system to respond differently. Examples are the operation of the sorting algorithm (which is now fairly simple and therefore cannot be extensively tested), long names of individuals or letters with accents. Another example is performance. On a table with 1,000 fictitious names that are numbered consecutively, the database management system might treat them differently from a table with 1,000 fictional names. The so-called index on a table may be differently constituted, which may be detrimental to performance.

Entering test data

There is a choice of three possibilities for the entering of test data:

1. Entering through regular system functions
2. Entering through separate front-end software

3. Use of production data

1. Entering through regular system functions
Entering test data through regular system functions has the disadvantage that those functions themselves have often not been exhaustively tested and that the data entered therefore need to be thoroughly checked. The advantage is that during the accumulation of the files, the regular functions are implicitly tested simultaneously and the consistency between the data is guaranteed. A condition, however, is that the input functions need to be delivered first. This should be agreed in advance with the supplier of the software.

2. Entering through separate front-end software
Entering test data through separate front-end software and test files has the risk that the test environment will contain inconsistent or non-permitted situations, since there was no check on the input. This means that technical support is required with the accumulation and, of course, tested front-end software must be available. The advantage is that the files can be accumulated relatively quickly.

In more detail

Working with 0 data, 0 scripts and 0 files

0 Data are test data that are initially required for the execution of the test. 0 Data can take many forms. It can consist of, for example, persons with a name, address and other features that are used in various test cases. It can also be the users who are permitted to use the system (the testers). Another form again is the data in so-called master tables. It is important to identify and describe the required 0 data in the specification of the test cases.

0 Scripts are test scripts with which the 0 data is placed in the system. This takes place via the regular system functions, with the advantage that the functions of the system concerned are already being tested. An added advantage is the clarity of the starting point/data (0 scripts are executed on an empty database). 0 Scripts are executed first, and therefore, with the execution, the tester can gain an initial impression of the quality of the test object.

A condition for working with 0 scripts is of course that the functions required for inputting 0 data are built first. If that is not the case, it may be decided to work with so-called **0 files**. These files contain the 0 data and can be read into the database direct via separate front-end software (e.g. based on SQL).

3. Use of production data
The use of production data as test data has the advantage that testing can be done with a lot of data, that the files can be built up quickly and that any conversion software is tested implicitly. A disadvantage is that these data show little variation and it can mean a lot of searching for the right variation in starting point data in a test case. Another disadvantage is that it is not always permitted to work with production data (because of privacy legislation or openness to fraud). This makes it necessary to make identifying data unrecognizable. In some cases, a production copy is not frozen for the test, but a new copy is periodically placed in the test environment. The disadvantage of this is that the tests are not directly repeatable, because the production data of each copy are different each time, so that the test result predictions are no longer correct.

Tip

A variant on obtaining test data from production is to have test data supplied by users. No one knows the system and associated data better than they do, including the 'difficult' cases. Ask each user for a number of difficult cases in the form of test data. This can be done by having the user himself take his place behind the test object and input these cases. Another possibility is to copy the specific cases from production and put them into the test environment.

Aside from planning and budget difficulties, the first alternative, entering test data through regular system functions, is preferable. If the test team has permission to obtain test files from production, it is also possible to combine the three alternatives. Choose a collection of production data that, for example, contain a particular type of information (customer, order, invoice, etc.). This subset is loaded into the test environment (retaining consistency among the various data). Subsequently, with the aid of regular system functions, changes are made to these data to create the desired starting point.

In more detail

Test data in data-warehouse testing
A data warehouse can be generally split into two groups of programs:

- The extraction and conversion programs for filling the data warehouse
- The reporting programs for obtaining information from the data warehouse.

While it is preferable to use separate test data for testing individual extraction and conversion programs, production data are inclined to be used with integral testing of the reporting programs. The reason for this is that the creation of a consistent set of fictional test data is demanding and with a set of production data, this consistency is almost automatically guaranteed. Besides, a big advantage is that a user can assess the outcome of a report more easily when using real production data.

Disadvantages of using production data in testing a data warehouse, however, are:

- The difficulty of making exact output predictions, since it is difficult to find out what the input was
- The confidentiality associated with some data. In practice, this means that the use of production data is not possible, or only after application of scrambling techniques (depersonalization, making data unrecognizable)
- The continually changing situation: the production data of today are different from those of a week ago, which hampers retesting.

This last disadvantage can be helped by suspending the daily/weekly reloading of data, so that the same starting point can always be used. An applied simplification is not taking the entire production files, but a selection of them. However, this requires focus (and time) for the mutual consistency of the data.

Delta test1
As an addition to this, the following procedure may be gone through:

- Take a subset of production data and call this X
- Run subset X in its entirety through the data warehouse and record the results
- Now add to subset X a number of self-created test cases and call this set X+1.
- Run subset X+1, too, in its entirety through the data warehouse
- The results of X+1 can be predicted by adding to the results of X the same self-created test cases
- Then add test cases to subset X+1 and call this X+2
- Run subset X+2 in its entirety through the data warehouse

- The results of X+2 can be predicted by adding the self-created test cases to the results of X+1 (this second run is useful for checking changes in time).

Delta test2
The following is a somewhat simpler variant of the above:

- Empty the database(s) of the delivering systems
- Put a number of self-specified test cases into these systems
- Run the extraction and check the result in the data warehouse
- Now put the same test cases as a kind of regression test into the full database(s) of the delivering systems
- Run the extraction and check the result (of the test cases) in the data warehouse.

Example

For a test process in a big data warehouse, the following test files are used as test data:

- The small test set: this is as small a test file as possible, with which the, possibly obvious, functional problems in the use of the prototype are searched for effectively. This test set is used as the first test after the development or reworking of the prototype and with all the other tests to quickly obtain an impression
- The 1,000-records test file is used for the functional acceptance test and consists of around 1,000 records from a daily file. The daily file that is used for this should concern a day in which as many (problem-generating) different cases occur. The choice of this is determined together with the client
- The 5% (or X%) test set is a representative sample compiled by the client from the source files for the third increment
- The 'daily files' test set consists of a complete daily file. The daily file that is used for this should concern a day in which as many (problem-generating) different cases as possible occur. The choice of this is determined together with the client. The execution of a weekly process in order to check whether starting status + mutations = final status is an important point of focus here. Preliminary dates of Wednesday 1 March, Thursday 2 March and Friday 3 March are used, after which a weekly process is run to check this.

Use of starting points during the test

The use of the central starting point during the test should be considered in advance. This chiefly concerns the choice between:

1. The cumulative construction of the central starting point (unstructured or structured)
2. Periodic restore with the central starting point (master copy)
3. The parallel use of several versions

1) The cumulative construction of the central starting point (unstructured or structured)
With cumulative construction, the central starting point grows along with the tests. If this is done in an unstructured way, the testers input new test data as required. This gives the testers much freedom and flexibility, but also has a disadvantage. A variety of testers input their own test data, which can influence the test results of other tests. This can cause a lot of wasted searching time in the analysis of test results. Besides, data will quickly become inconsistent.
With the structured variant, the testers make agreements in order to prevent such influences. For example, they may agree that only certain types of test data may be entered or changed, or that test data should be identified so that it can be seen to which tester they belong.

Example

For the testing of a mobile telephone subscription billing system, a test team of 5 persons was involved. Each of these individuals was responsible for the testing of a specific subsystem. In order to avoid the testers getting in each other's way when using the central starting point, it was proposed to link a range of telephone numbers to each subsystem. The starting point of the test cases for a specific subsystem then had to fall within that range. A range was also agreed for the integral test that ran across the various subsystems. This resulted in the following division:

Subsystem 1: range of telephone numbers +31610000000 to +31619999999
Subsystem 2: range of telephone numbers +31620000000 to +31629999999
Subsystem 3: range of telephone numbers +31630000000 to +31639999999
Subsystem 4: range of telephone numbers +31640000000 to +31649999999
Subsystem 5: range of telephone numbers +31650000000 to +31659999999
Integral: range of telephone numbers +31690000000 to +31699999999

2) The periodic restore with the central starting point (master copy)
A second approach is the regular restoring of the central starting point (also called the 'master copy'). This is done via a backup-and-restore procedure. A backup is first made of the master. At certain times, the administrator of the master restores it. That may be periodically, for example every day of the week, but also on request, for example after the execution of a test. A special management procedure can provide for the structurally adding of test data to the master. A big advantage is the manageability of the data, but disadvantages are the dependency of the restore point and the extra work to go from the master to the starting point necessary for the test.

3) The parallel use of several versions
A third possibility is the use of several environments with parallel versions of the data. Each tester has his own test environment and starting point(s). Having a central starting point at your disposal may remain useful, but each tester is able to amend it as he wishes in his own environment. A big advantage of this approach is the independence of the tests: disruption by other tests is barely possible, since the tester knows exactly what is in his own starting point. That delivers great savings in time. A disadvantage is that, because of the isolation of the tests, faults in starting points can remain undetected for long periods and integral test aspects are only dealt with at a late stage. Another disadvantage is the extra cost for the required test environments, both in terms of hardware and of administration. An important condition for this method of operation is good configuration management. This should ensure that the software deliveries and follow-up deliveries in connection with solved defects are rolled out to every test environment simultaneously. This could be a risk factor.

In more detail

Test environments and test data within SAP®
The terminology of SAP speaks of a system landscape, containing the various environments. A system landscape often consists of separate development, test, acceptance and production environments (also known as DTAP). These environments are called clients. There can be several clients per environment (instance) present. Several clients ensure that the testers do not get in each other's way as regards test data. It is advisable to set up a separate master client to secure the test data. Through copying, these data can be placed in another client. SAP also has the Test Data Migration Server tool, with which data from a productive environment can be reduced and if necessary anonimized and transferred to non-productive environments.

The transferring of changes (customizing, new software) in SAP from one environment to another is done by means of so-called transports (SAP Transport Management System). Transporting can be client-dependent or client-independent. With transporting, it is necessary to maintain a certain sequence and it is sometimes necessary to create certain settings manually per environment. All of this requires very good configuration management containing release or transport administration.

The figure below contains an schematic example of the environments and associated transports.

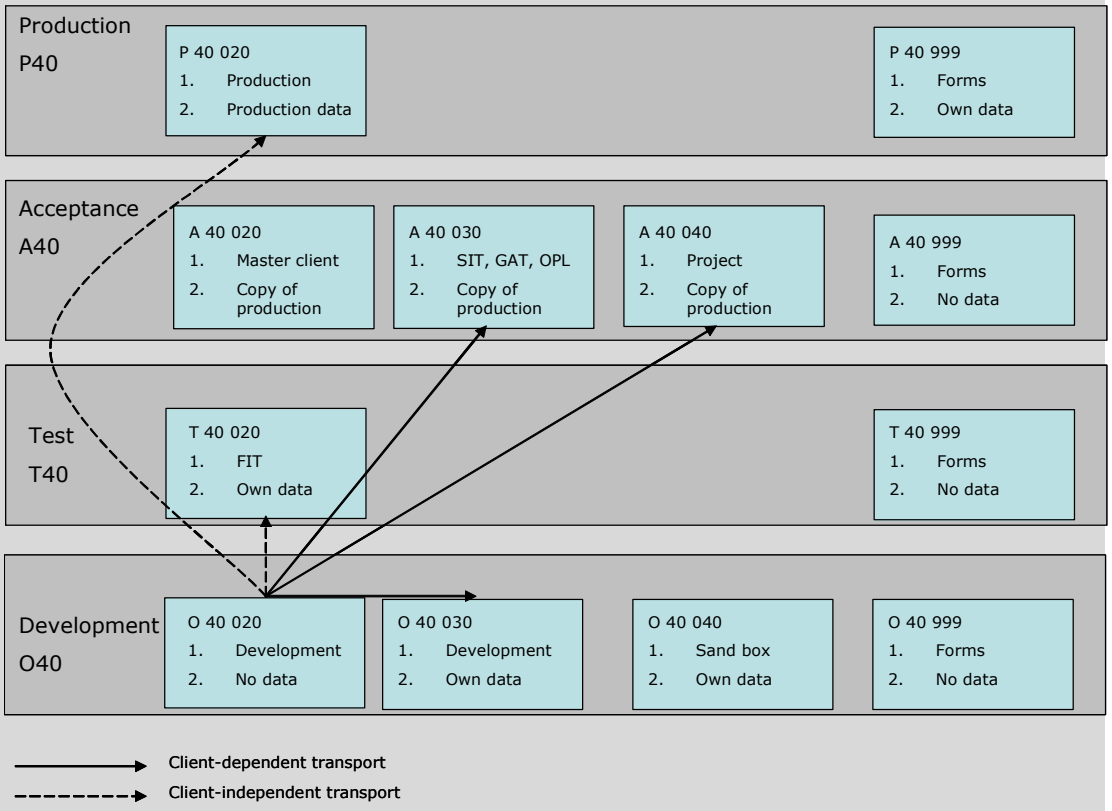


Figure 54. SAP environments and transports.

Test data with outsourcing

A development that is attracting the attention of various legislators is the handling of electronic data during outsourcing. Two subjects warrant special attention here:

- Confidentiality of the data used
  Increasingly, it is being established in laws or formal guidelines how electronic personal details should be dealt with and how to guarantee that such information remains confidential. When test data are created from production databases, it is necessary in cases of outsourcing that the data is made anonymous, since the data departs the organization and sometimes the country. Cases are known of employees of the supplier abusing software and data belonging to the outsourcing organization.

  Tip

  With anonimization, take care that all the data are anonimised in the same way, so that they remain consistent with each other in a variety of files.

- Responsibility for supply of data
  Another point of focus is the specifying of the test cases and the necessary 0 data. The supplier sometimes has insufficient subject knowledge to create realistic values himself. Extreme examples are: using postcode tables with a wrong number of numeric positions or setting the VAT percentage at 100%. This can seriously disrupt the execution of tests and also makes checking of the test results extremely difficult. If certain 0 data are important to a good test, agreements should be made concerning who will deliver them, and when.

Products

Test basis defects
A description of the central starting point(s)

Techniques

Not applicable.

Tools

Test data tool.

4.3.6.3 Specifying the test object intake

Aim

The preparation of the intake of the test object so that testing can start as soon as possible after delivery of the test object.

Method of operation

This activity contains the following subactivities:
- Creating a checklist for the test object intake
- Creating a pre-test test script.

Creating a checklist for the test object intake

At a certain point, the test team takes delivery of the test object. This first activity has the aim of establishing whether the delivery of the test object is complete, i.e. that it contains what was agreed with the supplier of the test object – no more and no less.
The test object usually consists of all kinds of software components (each with a particular version), but a user manual and installation guide, too, for example, may be part of it. The tester documents in the checklist which parts are expected with the delivery.
Besides information on the test object itself, the checklist also contains questions on the delivery information. It should be apparent which changes the delivery contains and which parts are related to which change. This prevents the test team from receiving changed software parts, while they have no change proposal and therefore no test planned for it.

In more detail

This occurs with specialist packages, in particular, because the supplier implements the change proposals of a number of customers simultaneously, but only provides feedback to each customer individually concerning the changes requested by them.

Creating a pre-test test script

After installation of the test object, a pre-test takes place in order to determine whether the test object is good enough to start testing. In this activity, the testers prepare this pre-test by creating a test script. This can involve several degrees of thoroughness. Below are a few examples:

- Checklist with all the functions, which should all be accessible
- For a number of representative functions, a simple test case with valid input ("good case") is specified
- Specification of test cases solely aimed at integration to check that the various parts can communicate with each other. The data-cycle test is a good choice for this.

The test cases may be obtained from the regular tests, but the results check is much more flexible. For example, it is not important for the pre-test that the test case delivers a correct result, as long as it delivers a result and does not crash, for example.

Examples

- For a banking application, the pre-test consists of a script of 25 end-to-end test cases.
- With another financial organization, the pre-test takes a day in which a tester executes the test cases which contain the most important functionality.
- A telecommunications organization requests the supplier of the software to execute a number of end-to-end test cases as a pre-test.

Products

Checklist of test object intake test
Pre-test test script.

Techniques

Not applicable.

Tools

Not applicable.

4.3.7 Execution phase

Aim

To obtain insight into the quality of the test object through the execution of the agreed tests.

Context

The actual testing takes place during this phase. The test object is delivered and as much as possible has been prepared in the preceding phases in order to keep the test execution as brief as possible.

Preconditions

The following conditions should be met before the Execution phase can commence:

- The test object, or a separately testable part of the test object, should have been delivered.
- The test scripts for the test object, or the separately testable part of the test object, should be ready.
- The intake of the associated test infrastructure should have been completed successfully.

Method of operation

The actual execution of the test begins at the point when the test object, or a separately testable part of the test object, is delivered. The test object is first checked for completeness. Subsequently, it is installed in the test environment to assess whether it functions as it should. This is done by carrying out a preliminary test, the so-called pre-test. This is a general test, with the aim of examining whether the information system under test, in conjunction with the test infrastructure, is of sufficient quality to undergo extensive testing. If, on the whole, it is of sufficient quality, the central starting point is prepared. The test may be executed on the basis of the (manual or automated) test scripts that were created in the Specification phase. In that case, the starting points for the test scripts should first be prepared. Execution of the test can also be carried out in an exploratory manner, or on the basis of checklists. During the execution, the test results are logged. Investigation of the causes of any differences between expectations and obtained test results takes place after the test execution. Causes of differences may lie in software faults, but other causes are possible. For example, there could be mistakes in the test basis, in the test environment or in the test cases. When a fault has to be solved, this is formally reported as a defect. When the defect has been solved, a new test can be executed.

Thus, this phase often involves an iterative process of test-rework-retest. The substance of this iterative process depends on the cause of the fault. For example, a fault in the test basis can result in a renewed (re)planning of the test, after which the phases of Preparation, Specification and Execution are gone through again. With a fault in the software, the iterative process of test-rework-retest may be restricted to a repeat of the Execution phase.

Roles / responsibilities

All the activities can be carried out by all the test-team members. However, the check on completeness of the test object is done by the test manager, aided by the (if applicable) test infrastructure coordinator.

Activities

Within the Execution phase, the following activities are distinguished:

1. Intake of the test object
2. Preparing the starting points
3. Executing the (re)tests
4. Checking and assessing the test results.

The following scheme shows the sequence and dependencies between the various activities.

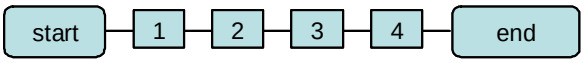


Figure 55. Execution Phase

4.3.7.1 Intake of the test object

Aim

To establish whether the delivered parts of the test object function in such a way that adequate testing can be carried out.

Method of operation

The method of operation includes the following subactivities:

- Checking completeness of the delivered test object
- Executing the pre-test.

Checking completeness of the delivered test object

With the aid of the checklist created in the Specification phase, the delivered test object is checked for completeness. This is done by the test manager, assisted by (where this role is taken) the test infrastructure coordinator. Missing parts are reported, by means of a defect, to the parties involved. If the defect is test-obstructive (i.e. the subsequent subactivity, the pre-test, cannot start), then this should be solved immediately. It is advisable to carry out this subactivity together with the department that maintains the test environment, since this department depends on a complete delivery of the test object, otherwise the installation will be wrong. Moreover, they have the technical knowledge to be able to check the test object on the aspect of completeness. Following approval, the test infrastructure coordinator can install the test object (or have the administrators do so).

Executing the pre-test

As soon as a (version of the) test object is installed, it is important to carry out a pre-test. This takes place before the actual testing begins. The purpose of the pre-test is to evaluate whether the test object is of sufficient quality for testing. The pre-test is carried out by executing the test script that was created for this during the Specification phase. It regularly happens in practice that systems are wrongly delivered or wrongly installed in the first days of testing, thus delaying the start of the test execution. This is not only a waste of time, it also demotivates the test team. It is important to consider this when creating the test plan.

Tip

The pre-test as negotiation reinforcement
With a pre-test, the test manager's position is strengthened considerably when he wants to argue that the clock has not yet started for the main test. That is to say, if from Monday, 10 days of test execution were planned and the pre-test only succeeds on Wednesday, the 10 days only begin from Wednesday. There is room for discussion here, but the test manager will have a much stronger negotiating position.

A condition of the execution of these subactivities is that the required test infrastructure is available. This means effectively that the intake of the infrastructure should have gone successfully (see section 6.4.4 "Intake of the infrastructure"). A successful pre-test is a condition for the starting of the subsequent activities in the Execution phase. The defects from the pre-test are registered and, if a defect is test-obstructive, it is immediately submitted to the parties involved. Every effort should be made, with the highest priority, to solve the test-obstructive defects and allow the pre-test to complete successfully.

Example 1

With the building and testing of a new registration system, time is running out. It is decided to ask the testers to sacrifice their free Saturday and Sunday and to test over the weekend. Experience shows that the delivery and installation of the test object in the test environment does not always progress smoothly. In many cases, parts of the test object are missing or do not work at all.

In order to avoid the testers turning up on Saturday for nothing because the installation of the test object has failed, it is decided on Friday evening to carry out a pre-test. This is done by the test manager and the test infrastructure coordinator. Together, they check at 18:00 hours, on the basis of a selection of test scripts, the quality of the delivered test object. If this is sufficient for testing, they contact the test team members before 20:00 hours to tell them that the test is going ahead.

Example 2

A new version of an administrative system has been developed by an external supplier on the other side of the world. It is agreed with the supplier that, before it is delivered and the FAT is carried out, a pre-test will take place. This test is carried out, via the Internet, on the supplier's infrastructure. It provides an initial impression of the quality of the system and confirms that the FAT can actually begin. This also avoids the installation of the test object in the company's own infrastructure causing problems through distrust of the supplier. They have witnessed it working there, after all!

Products

Defects
Installed and testable test object.

Techniques

Not applicable.

Tools

Testware management tool
Defect management tool
Automated test execution tool
Monitoring tool
Comparator
Database manipulation tool
Simulator
Stubs and drivers.

4.3.7.2 Preparing the starting points

Aim

To prepare the starting point required for the execution of the tests.

Method of operation

Before the execution of the test cases in the test script can begin, the test object should be placed in the appropriate condition or situation. This not only involves the preparation of the test data necessary for the processing, but also the setting of the system and test environment in a particular condition. It may concern, for example, the formatting of a disk, or even the configuring of an input device.

Two types of situations of the test object are distinguished within TMap:

1. A *central starting point* for a number of tests
2. A *starting point* per test script.

At the start of a test, the central starting point is created. The test object and test environment are then ready to receive the input in accordance with the test scripts (at any rate those that are executed first). The test data are gathered as described during the activity "Defining central starting point(s)" in the Specification phase. The gathering of these test data can take place in various ways. Defects found during the gathering of the test data are registered in accordance with the procedures laid down in the test plan.

Tip

Backing up the central starting point and checking this
As soon as the test object has been placed at the central starting point and checked, it is advisable to create a backup. This can be restored at any given point. It is important to carefully check this principle of backup and restore before commencement of the tests.

Products

Defects

Central starting point
Starting points.

Techniques

Not applicable.

Tools

Testware management tool
Test data tool
Model-based testing tool
Automated test execution tool
Performance, load and stress test tool
Database manipulation tool.

4.3.7.3 Executing the (re)tests

Aim

To obtain test results, on the basis of which evaluation of the test object can take place.

Method of operation

The method of operation includes the following subactivities:

- Executing explicit tests;
- Executing implicit tests;
- Executing evaluations on end products.

Executing explicit tests

In explicit testing, explicit test cases are executed to obtain information on the property (quality characteristic) or system part under test. Results are obtained by running software and executing operations on the test object. These results are compared in the subsequent activity against the expected results, thus delivering any defects. Explicit testing is the most usual way of testing. There are two possible types of explicit testing:

- Testing on the basis of specified tests created in the Specification phase.
  The specified tests that are created in the Specification phase form the starting point for the tests to be executed here. These may be test scripts containing the test actions and checks or the physical test cases. The test scripts are described in an optimal sequence and form the stepped plan for the test execution. If it has been decided to use tools for automated test execution, then the specified tests are executed with the aid of a test tool. In addition, tests can also take place on the basis of checklists or in another form. An important condition for a worthwhile explicit test is that the testers do not deviate from the test cases and execute at least the described test cases. Otherwise, there is no way of guaranteeing that the strategy laid down in the test plan is actually being carried out.

- Testing on the basis of an exploratory technique.
  With this type, the tester carries out exploratory work during the explicit test. This means that the tester is examining the application under test piece by piece, thinking about what should be or could be tested (test design) and then does it (test execution). In doing so, the tester is gaining knowledge of the application, considering what should be tested next, testing it, et cetera. The design and subsequent execution of the tests

take place in close succession. Possible techniques are "Exploratory Testing" and "Error Guessing".

Tip

A quick way of helping inexperienced testers on their way is to carry out this activity in pairs. Team up an inexperienced tester with an experienced one [Kaner, 2001]. In this, one tester is responsible for the test. He involves another tester, with one of them operating the keys and the other thinking about the things to be tested, observing, taking notes and researching. By thinking aloud, the testers together generate many more ideas than they would separately. They also help each other not to lose sight of the test goal because of unimportant details. Coaching in pairs is certainly to be recommended, particularly in the beginning. Testing in pairs is less successful if the individuals are very introverted or very assertive.

These two types of explicit testing do not exclude each other. In fact, when applied in combination, they can reinforce each other. Reasons for combining the two types may be:

- During the execution of the specified tests, it is felt that insufficient insight into the quality has been obtained. By now testing exploratory in a number of areas within the test object, this impression can be either substantiated or dispelled.
- The strategy for a retest may be that only those parts of the test object are tested that have been amended by the programmers. In order to be sure that the unchanged parts still work, they can be subjected to some extra, exploratory, testing.
- The addition of exploratory testing over and above the specified testing can be useful as a stimulus for creative testing. This could be scheduled, for example, for a Friday afternoon. Many testers are more creative during this part of the week. Just before the weekend, the mood is good and everyone is open to experimentation. These experiments may cover very exceptional situations, but perhaps also those that are so ordinary that they are overlooked. It is then that crucial faults may be found in the test object.
- During the execution of a test script, a fault may occur. This has to be investigated further, before it is reported as a defect. It can be observed whether the defect always occurs or only in the specific situation. Alternatively, perhaps the defect occurs in other (similar) areas in the test object. There is also the possibility that several defects are located together. This investigation can take place on the basis of exploratory testing (see also section 4.7 "Defects management").

Tip

Faults located together
Faults have a tendency to clustered together within a test object. If a fault occurs in a particular function, screen, operation or other part, the chances are that other faults are there as well. There are various causes for this. For example, the particular part may contain more complex code, so the likelihood of the programmer making a mistake is greater. Alternatively, a particular part may have been created by an inexperienced programmer, or by one who was having an off day. It is therefore advisable, when a fault is found, always to search the area for other faults.

Executing implicit tests

During explicit testing, information can also be gathered on other properties (quality characteristics). No explicit test cases are designed for these. This is referred to as implicit testing, and the tests can be executed planned or unplanned. If planned, it is agreed in

advance that this is to form an actual part of the test strategy. Testers can then be asked in advance of the test execution to observe a number of characteristics (such as performance or usability) of the test object. This is therefore not based on any targeted test cases. Another way is to question the testers after the execution of the explicit test. However, there is the risk that, since no specific attention has been paid to these things, wrong information will be given.

Unplanned implicit testing arises because, during execution of the test, certain things start to catch the attention. It is agreed to observe them more closely. If, for example, regular system breakdown takes place, a decision can be taken as regards reliability. Alternatively, if certain screens do not have an appealing look and feel, something can be said about the usability.

Executing evaluations

It is laid down in the test strategy whether evaluation on end products should be carried out. In evaluations, products are assessed without any software being run. This evaluation usually consists of the inspection of documentation, such as security procedures, training, manuals, et cetera and is often aided by checklists. On the basis of these, it is attempted to obtain insight into the relevant quality aspect. Here too, any defects are registered and processed by means of the defects procedure (see section 4.7 "Defects management").

Products

Test results.

Techniques

Exploratory Testing
Error Guessing.

Tools

Testware management tool
Defect management tool
Model-based testing tool
Automated test execution tool
Performance, load and stress test tool
Monitoring tool
Code coverage tool
Comparator
Database manipulation tool
Simulator
Stubs and drivers.

4.3.7.4 Checking and assessing the test results

Aim

To analyze the differences between the obtained test results and the predicted results in the test scripts or checklists.

Method of operation

The method of operation includes the following subactivities:

- Comparing test results
- Analyzing differences
- Determining retests.

Comparing test results

The test results are compared against the predicted results in the test scripts and checklists. If testing is being done based on an exploratory technique, the tester will compare the outcome against the documented test basis, such as the functional design or a requirements document. If there is no documented test basis, the tester needs to find other ways of comparing the outcome. This information can be obtained, for example, from norms and standards, memos, user manuals, interviews, advertisements or rival products.

In more detail

The dangers of testing without a documented test basis
If no documented test basis is available to the tester, there is a real risk that he or she will begin to rely on information sources other than the test basis, such as his or her intuition. An unwanted end result may be that system and documentation are running out of sync. If the system is correct and the documentation wrong, this can lead to maintenance or administration problems. Conversely, it is possible that (deep) functionality is described in the documentation that has been incorrectly implemented in the system and that has not emerged with testing based on sources other than the system documentation. Another unwanted end result may be that, in the absence of clarity concerning the scope, the testers generate an endless stream of change requests in the form of defects.

If there are no deviations, this is logged. If deviations are found, they are analyzed. The comparing of the test results often takes place simultaneously with the execution of the test. For example, by checking off the steps in the test script it can be indicated whether a test result corresponds with the expected result. In certain cases, it is not possible to do this during the test (e.g. with batch systems, where the output of several test cases is presented).

Analyzing differences

The differences found are further analyzed during this subactivity. The tester should perform the following steps:

- Gather evidence
- Reproduce the defect
- Check for own mistakes
- Determine suspected external cause
- Isolate the cause (optional)
- Generalize the defect
- Compare with other defects
- Write defect report
- Have it reviewed.

These steps are explained in the section on "Finding a defect". The steps are listed in the general sequence of execution, but it is entirely possible to carry out particular steps in another order or in parallel. If, for example, the tester immediately sees that the defect was already found in the same test, the interim steps need not be performed.

In the test scripts, the numbers of the defects are registered with those test cases where the defect was found. In that way, it quickly becomes clear in any retest at least which test actions need to be carried out again. Various test tools are available both for comparing the test results and for analyzing the differences.

Determining retests

Reasons for carrying out retests may be found defects. If the cause of the defect concerns a fault in the test execution, the relevant test is carried out again. Defects that have their origin in a wrong test script or checklist are solved. Thereafter, the changed part of the test script is executed again or the entire checklist is gone through again. Faults in the test environment should also be solved, after which the relevant test scripts are executed again in their entirety.

Faults in the test object or the test basis will usually mean a new version of the test object. With an fault in the test basis, the associated test scripts will usually also need to be amended. This often involves a lot of work. When retests take place, it is important to establish the way in which they are to be carried out. The test manager will determine in the Control phase whether all the test scripts should be carried out again in whole or in part, and this partly depends on:

- The exit criteria set out in the test plan
- The severity of the defects
- The number of defects
- The degree to which the earlier execution of the test script was disrupted by the defects
- The time available
- The risks.

In more detail

When to test solved defects?

Defects that have been solved must be tested again. The timing of these tests can be quite different.

1. Test as soon as a defect is solved. The advantage of this is that the programmer, who has solved the defect, still has it fresh in his memory. He can therefore act quickly in the event that the defect appears not to be solved. The disadvantage is that the code is often changed, delivered and tested. Mistakes can be easily made here, and that is less efficient for the tester.

2. Gather solved defects and test these. The advantage of this is that defects can be solved and tested collectively (e.g. per module or per screen), which is a more efficient way of working. The code is also more stable, so that the chances of a defect returning are minimal. The disadvantage, however, is that this method takes longer.

The choice of option 1 or 2 depends on the project and the way of working. If it is possible to deliver a release of an application every day (also known as the 'daily build') and there are a large number of defects to be retested, the strategy may be to choose a mix of the above. It is then determined each day which solved defects will be included in the release and these can then be tested by the test team the following day. It is important in that case to set up a separate test environment and only to use it for testing the solved defects in the releases. In addition, a test of the entire test object will have to take place at the end, in order to establish that nothing else has changed (regression).

Products

Defects
Logging of the test results.

Techniques

Not applicable.

Tools

Testware management tool
Defect management tool
Test data tool
Automated test execution tool
Performance, load and stress test tool
Monitoring tool
Code coverage tool
Comparator
Database manipulation tool.

4.3.8 Completion phase

Aim

- To learn from experience gained during this test
- To preserve testware for reuse in a future test.

Context

With the structured test method of TMap, much benefit is to be gained from the possibility of repeating the process. This allows products, provided that they meet certain requirements, to be reused in a subsequent test. In turn, this can ensure that certain activities will proceed faster. Products may be tangible things, such as test cases or test environments, but also intangible things, such as valuable experience.

Preconditions

The following condition should be met before the Completion phase can commence:

- The test execution is almost finished.

Method of operation

The test process is evaluated. The aim here is to learn from the experience gained and to apply the points of learning to any new test. This also serves as input for the final report, which the test manager creates in the Control phase. Also a selection is made from the often large quantity of testware, such as the test cases and the description of the test infrastructure. The point here is that with changes and associated maintenance tests, the testware only requires adjustment, so that it is not necessary to design a completely new test. During the test process, efforts are made to keep the test cases corresponding with the test basis and the developed system. If necessary, the selected test cases should be updated.

Roles / responsibilities

All the activities can be carried out by all the team members.

Activities

The Completion phase consists of the following activities:

1. Evaluating the test process
2. Preserving the testware.

The scheme below shows the sequence and dependencies between the various activities:

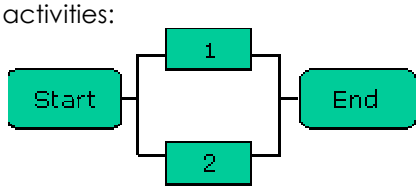


Figure 56. Completion phase

4.3.8.1 Evaluating the test process

Aim

To learn from experience gained during the completed test and to document the learning points for future tests.

Method of operation

Continuous learning, followed by using the new knowledge, is an important topic in TMap. A way of doing this is to organize evaluation sessions. These sessions are mostly aimed at generating lessons and learning experiences for the future. The subject of evaluation may vary, according to requirements. It may concern the evaluation of the test process, the results of the test, the involvement of the various parties in it, the use of the test infrastructure et cetera. It is important here to clarify how the people involved in the test experience the subject. An evaluation should take place upon completion of the test, but it is advisable also to do this regularly during the test itself. In this way, it is possible to learn continuously and to apply what has been learned. An aid for asking the appropriate questions during an evaluation is the "Evaluation of the test process" checklist (www.tmap.net).

The test manager creates a final report in the Control phase. This report describes how the test process has performed. It also supplies figures for purposes of future test processes, and the result of the evaluation serves as input here.

Tip

Evaluations as leverage for change
The carrying out of evaluations may have a purpose that extends beyond simply reusing the acquired knowledge. It may also have the purpose of setting up the knowledge as leverage for change. A condition for this is that the test manager's role is one in which he can propose changes. These proposals can then be included in the final report. If this process is already taking place during the execution of the project, there is the great advantage that the changes can be implemented immediately. A condition for success is that the sharing of knowledge be encouraged at every level, this is possible by, for example, organizing (informal) meetings. During these meetings, there should be a relaxed and egalitarian atmosphere. Involve all the parties in the meeting, talk about the problems and try to find immediate solutions.

Example

Testers as sounding board
During a test, more and more English-speaking developers came to the Dutch-speaking testers to ask questions concerning particular functional specifications. It appeared from various informal meetings on Friday afternoons that they had difficulty with the combination of the broken English of the Dutch designers and the long screeds of text. From within their expertise, the testers had built up in-depth knowledge of the system. In consultation with the various parties, it was then decided that the testers could serve the developers as a sounding board. In addition, a selection was made from existing test scripts that had to be executed by the developers before they delivered their piece of software. This benefited the general quality and pace of the test process, defects were now found during development and there was a greater involvement of the developers in the testing and vice versa.

Products

Evaluation of the test process.

Techniques

"Evaluation of the test process" checklist (www.tmap.net).

Tools

Testware management tool
Defect management tool.

4.3.8.2 Preserving the testware

Aim

To select and update the produced testware in such a way that optimal use can be made of it in future tests.

Method of operation

The method of operation includes the following subactivities:

- Selecting testware
- Collecting and refining testware, making it accessible
- Transferring testware.

This activity has a close connection with the activity "Preserving the infrastructure" in the Setting up and maintaining infrastructure phase.

Tip

Starting the activity of preserving testware earlier
Although Preserving Testware is the last activity in the TMap life cycle model, it is advisable to start this early on. By allowing for the possibilities of preserving testware as early as the Specification phase, certain standards can be developed or certain tools can be employed, so that the eventual preservation will proceed faster. By, for example, working from the start with consistent version-numbering and a central store for all the products, there is no need to search for the latest version of all the products during this activity. The use of a testware management tool can help with this. How the preserving takes place is defined in the activity "Organizing the management" in the Planning phase.

Selecting testware

In consultation with the future administrator of the system, an inventory is drawn up of which testware is to be made available to him. The purpose is to render the testware reusable for changes and associated maintenance tests, so that it will not be necessary to design a completely new test. The final choice of testware to be made available is made on the basis of a costs/benefits analysis. Subjects in this would be 'What will it cost to maintain the testware (storage and updating)' and 'What will it cost to make it a new'.

The test products to be delivered are set out in an inventory. This is an overview of the test products to be preserved. It is important to indicate the way in which the test products were created, in order to facilitate appropriate future maintenance. Bear in mind here in particular the test design techniques, tools, et cetera that were used.

Collecting and refining testware, making it accessible

The testware to be transferred should be completed and adjusted where necessary. During the last phase of the execution, in particular, maintenance of the testware is often postponed. Before transfer to the future users can take place, any changes should be processed. The testware should also be made accessible. This means that it should be stored in such a way that it is readily available to the future users. That may mean, for example, that the directory structure has to be set up differently or a particular tool must be used.

In more detail

Adjust regression test set
The updating of a regression test set is often overlooked. For that reason, it is advisable to include this activity as standard within the activity of "Preserving the testware". During the test execution, it may have been that the system reacted differently from what was assumed in the test script. If this is the case, the test script should be amended in accordance with the new situation. It should also be determined whether, and if so which, new test scripts need to be added to the existing regression test set.

Transfer of testware

Finally, the actual transfer of the testware takes place. In accordance with the testware inventory, all the selected parts are electronically, and sometimes also physically (on paper), transferred to the maintenance department.

Products

Testware inventory
Reusable testware.

Techniques

Not applicable.

Tools

Testware management tool
Test data tool.

4.4 Quality characteristics

For purposes of testing, TMap employs the set of quality characteristics shown below. Another common set of quality characteristics can be found in the international standard ISO25010. The use of a set of quality characteristics, whether from TMap or from ISO25010, is recommended as a way to check for completeness. It allows you to check that, out of all the aspects or characteristics of a system or package under test, a careful decision has been made about whether or not to test these. It makes little difference which set is applied. Often, the organization has already made a choice. An illustration of the TMap quality characteristics comparable to ISO25010 can be found at www.tmap.net.

In more detail

There are a number of reasons for keeping to the TMap set of quality characteristics and not changing to ISO25010:

- In many organizations, TMap is the standard for testing, including the TMap set of quality characteristics. These organizations see little need to change over to another set of quality characteristics
- The testing of functionality is one of the most important areas of focus in testing, and is discussed a lot in this book. ISO25010 sees functionality as an umbrella concept, which takes in, for example, security and suitability. Therefore, within ISO, the testing of security and suitability fall under the testing of functionality. This is confusing in a book on testing

- ISO25010 is not necessarily better or worse than the TMap set; it is simply different
- While ISO25010 is an international standard, in practice it appears that many organizations make their own little variant on this, which detracts from the authority of ISO25010 as a standard. Various organizations also follow old versions of ISO25010 (e.g. ISO9126).

The quality characteristics distinguished by TMap:
- Connectivity
- Continuity
- Data controllability
- Effectivity
- Efficiency
- Flexibility
- Functionality
- (Suitability of) infrastructure
- Maintainability
- Manageability
- Performance
- Portability
- Reusability
- Security
- Suitability
- Testability
- User-friendliness.

A description of each quality characteristic is given below, with an indication of the ways in which the testing of these takes place in practice.

Connectivity

The ease with which an interface can be created with another information system or within the information system, and can be changed.

Connectivity is evaluated by assessing the relevant measures (such as standardization) with the aid of a checklist. The evaluation of connectivity therefore concerns the evaluation of the ease with which a (new) interface can be set up or changed, and not the testing of whether an interface operates correctly. The latter is normally part of the testing of functionality.

Continuity

The certainty that the information system will continue without disruption, i.e. that it can be resumed within a reasonable time, even after a serious breakdown.

The continuity quality characteristic can be split into characteristics that can be applied in sequence, in the event of increasing disruption of the information system:
- Reliability: the degree to which the information system remains free of breakdowns
- Robustness: the degree to which the information system can simply proceed after the breakdown has been rectified
- Recoverability: the ease and speed with which the information system can be resumed following a breakdown
- Degradation factor: the ease with which the core of the information system can proceed after a part has shut down
- Fail-over possibilities: the ease with which (a part of) the information system can be continued at another location.

Continuity can be evaluated by assessing the existence and setup of measures in the context of continuity on the basis of a checklist. Implicit testing is possible through the collecting of statistics during the execution of other tests. The simulation of long-term system usage (reliability) or the simulation of breakdown (robustness, recoverability, degradation and fail-over) are explicit tests.

Data controllability

The ease with which the accuracy and completeness of the information can be verified (over time).

Common means employed in this connection are checksums, crosschecks and audit trails. Verifiability can be evaluated, focusing on the setup of the relevant measures with the aid of a checklist, and can be explicitly tested focusing on the implementation of the relevant measure in the system.

Effectivity

The degree to which the information system is tailored to the organization and the profile of the end users for whom it is intended, as well as the degree to which the information system contributes to the achievement of the company goals.

A usable information system increases the efficiency of the business processes. Will a new system function in practice, or not? Only the users' organization can answer that question. During (user) acceptance tests, this aspect is usually (implicitly) included. If the aspect of usability is explicitly recognized in the test strategy, a test type can be organized for it: the business simulation. During a business simulation, a random group of potential users tests the usability aspects of the product in an environment that approximates as far as possible the "real-life" environment in which they plan to use the system: the simulated production environment. The test takes place based on a number of practical exercises or test scripts. In practice, the testing of usability is often combined with the testing of user-friendliness within the test type of usability.

Efficiency

The relationship between the performance level of the system (expressed in the transaction volume and the total speed) and the volume of resources (CPU cycles, I/O time, memory and network usage, etc.) used for these.

Economy is explicitly tested with the aid of tools that measure the resource usage and/or implicitly by the accumulation of statistics (by those same tools) during the execution of functionality tests. This aspect is often particularly evident with embedded systems.

Flexibility

The degree to which the user is able to introduce enhancements or variations on the information system without amending the software.

In other words, the degree to which the system can be amended by the user organization, without being dependent on the IT department for maintenance. Flexibility is evaluated by assessing the relevant measures with the aid of a checklist. Explicit testing can take place during the (users) acceptance test, by having the user create, for example, a new mortgage variant (in the case of mortgages) or (in the case of credit cards), change the way of calculating the commission, by changing the parameters in both cases. It is often tested in this way first, before the change is actually implemented in production.

Functionality

The degree of certainty that the system processes the information accurately and completely.

The quality characteristic of functionality can be split into the characteristics of accuracy and completeness:

- Accuracy: the degree to which the system correctly processes the supplied input and mutations according to the specifications into consistent data collections and output
- Completeness: the certainty that all of the input and mutations are being processed by the system.

With testing, meeting the specified functionality is often the most important criterion for acceptance of the information system. Using various techniques, the functional operation can be explicitly tested.

(Suitability of) Infrastructure

The appropriateness of the hardware, the network, the system software, the DBMS and the (technical) architecture in a general sense to the relevant application and the degree to which these infrastructure elements interconnect.

The testing of this aspect can be done in various ways. The tester's expertise as related to the infrastructural elements concerned is very important here.

Maintainability

The ease with which the information system can be adapted to new requirements of the user, to the changing external environment, or in order to correct faults.

Insight into the maintainability is obtained, for example, by registering the average effort (in the number of hours) required to solve a fault or by registering the average duration of repair (Mean Time to Repair (MTTR)). Maintainability is also tested by assessing the internal quality of the information system (including associated system documentation) with the aid of a checklist. Insight into the structuredness of the software (an aspect of maintainability) is obtained by carrying out evaluations, preferably supported by code analysis tools.

Manageability

The ease with which the information system can be placed and maintained in an operational condition.

Manageability is primarily aimed at technical system administration. The ease of installation of the information system is part of this characteristic. It can evaluated by assessing the existence of measures and instruments that simplify or facilitate system management. Testing of system management takes place by, for example, carrying out an installation test and by carrying out the administration procedures (such as backup and recovery) in the test environment.

Performance

The speed with which the information system handles interactive and batch transactions.

Portability

The diversity of the hardware and software platform on which the information system can run, and the ease with which the system can be transferred from one environment to another.

Reusability

The degree to which parts of the information system, or of the design, can be used again for the development of other applications.

If the system is to a large extent based on reusable modules, this also benefits the maintainability. Reusability is evaluated through assessing the information system and/or the design with the aid of a checklist.

Security

The certainty that consultation or mutation of the data can only be performed by those persons who are authorized to do so.

Suitability

The degree to which the manual procedures and the automated information system interconnect, and the workability of these manual procedures for the organization.

In the testing of suitability, the aspect of timeliness is also often included. Timeliness is defined as the degree to which the information becomes available in time to take the measures for which that information was intended. Suitability is explicitly tested with the aid of the process cycle test.

Testability

The ease and speed with which the functionality and performance level of the system (after each adjustment) can be tested.

Testability in this case concerns the total information system. The quality of the system documentation greatly influences the testability of the system. This is evaluated with the aid of the "testability review" checklist during the Preparation phase. Also for the measuring of the testability of the information system a checklist can be used. Things that (strongly) benefit the testability are:

- Good system documentation
- Having an (automated) regression test and other testware
- The ease with which interim results of the system can be made visible, assessed and even manipulated
- Various test-environment aspects, such as representativeness and an adjustable system date for purposes of time travel.

User-friendliness

The ease of operation of the system by the end users.

Often, this general definition is split into: the ease with which the end user can learn to handle the information system, and the ease with which trained users can handle the information system. It is difficult to establish an objective and workable unit of measurement for user-friendliness. However, it is often possible to give a (subjective) opinion couched in general terms concerning this aspect. User-friendliness is tested within the test type of Usability.

4.4.1 Test types

In this section, a number of specific test types are discussed. Apart from the regression test, this concerns test types for quality characteristics other than functionality. The reason for this is that these test types are becoming more common in practice, but preparation, specification and execution of these tests demand different types of knowledge than is the case with the functional test types. Per test type, explanation is given of the aspect the test type is aimed at, the relationship with the quality characteristics previously described, the significance of the test and what test techniques are possible.
The following test types are discussed in turn:

- Regression
- Usability

4.4.1.1 Regression

What is regression?

A system or package is more or less always subject to changes. When it is in production, its owner will want to implement certain changes or extensions. But amendments are made even earlier, when a system is being built or a package is being implemented. This usually relates to solved defects or implemented change proposals. With iterative or agile development methods, repeated issue of new, expanded releases (also known as increments) is even inherent in the method.
With the making of the amendments (or extensions), it is possible for mistakes to be introduced into unchanged parts of the system (or package), causing the quality to deteriorate. This phenomenon of quality deterioration is called regression, and it is the reason that unchanged parts of the system also need to be tested. Although regression can relate to all the quality characteristics, the testing of it in practice is aimed primarily at functionality.

Definition Regression is the phenomenon that the quality of a system deteriorates as a whole as a result of individual amendments.

Importance of regression testing

The chance that faults have crept into an unchanged part of the system following an amendment is smaller than if the part were to be newly built. Assuming that the risk is determined by damage x chance of failure, the testing of the unchanged parts of the system can take place with less testing effort than with a new or changed part of the system. However, this is not to say that the regression test demands little effort. In maintenance situations, in particular, the total effort for this regression test is often greater than the testing effort required for the detailed testing of the changes. The reason for this is that with maintenance, usually only a very limited number of functions change.

Definition A regression test is aimed at verifying that all the unchanged parts of a system still function correctly after the implementation of a change.

A good regression test is invaluable. Certainly in the maintenance situation, the test offers reassurance that the new version of the system or package still operates correctly as a whole.

Test design techniques

There are no prescribed fixed test design techniques for regression testing. All the existing techniques can be used to specify the test cases in the test. However, a regression test focuses mainly on the correlation between the parts of the system, since this is where the chances of regression are the greatest. This means that integration test cases and 'good path' test cases are preferable to test cases for exceptional fault-handling situations. The regression test is often initially stocked with test cases from the testing of new parts or the original new-build tests and later supplemented with test cases for testing changes. Suitable test design techniques are, for example, the data combination test, data cycle test and the process cycle test. If the product risk analysis is available for the new build, the damage factors assigned to characteristics and object parts can play a role in the constitution of this regression test. Either a limited or a full regression test can be carried out, depending on the risks and on the required test effort. For an explanation of the scalable regression test, refer to the section on "Specification Phase".

The regression test is sustained by adjusting or extending the test set on the basis of changes to the system, including both functional adjustments and solved faults. This keeps the regression test continuously up to date.

Because the regression test focuses on the system as a whole, the test is executed frequently (at least once for each release), while the test rarely changes very substantively. This is in contrast to a test for validating a specific amendment – usually only carried out for the release concerned. The combination of a high frequency of use and high level of stability means that a good level of reusability of the test is very important. It is therefore essential to create and maintain a well-structured and documented test set.

In the execution of regression tests, test tools used for automated test execution come into their own. The big advantage of the automated regression test is that, for little effort, the full test can be carried out every time and no choices have to be made as to which part of the regression test will or will not be executed.

4.4.1.2 Usability

What is usability?

As with most IT definitions, there is a variety to be found relating to usability. Even the International Standards Organisation has two definitions:

Definitions
ISO 9241-11: the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.
ISO/IEC 9126: a set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users.

From the various definitions, a number of aspects emerge that play a role in usability:

- Effectiveness
  Are users able to complete their task and achieve their goal with the system?
- Efficiency
  How much trouble and time does it cost users to do this?

- Satisfaction
  What do the users think of the ease of operation of the system?
- Ease of understanding
  How easily does the user understand what the system expects him to input, and how understandable is the output to him?
- Ease of learning
  How quick and easy is it to learn and remember how to operate the system?
- Attractiveness
  How attractive does the user find the system, as regards e.g. layout, use of color, graphics, film clips and interaction?
- Robustness
  How easily can the users make mistakes in the system; how serious are these, and how easily can they be rectified?

Who the abovementioned user is, and which tasks he wants to carry out, plays an important part. Users may be customers of the organization or users of the system within the organization, but this also includes e.g. system administrators. A distinction should also be made between untrained and inexperienced users as against trained and experienced ones, and about the context in which the system is being used. A web application on a smart phone has other standards for usability than a web application on the PC.

The TMap quality characteristics that have most to do with usability are user-friendliness and effectivity. In respect of the latter characteristic, usability testing looks at the effectivity from the user's standpoint, not at the general effectivity for the organization in total. Usability testing also has some overlap with characteristics such as performance (if the system is not fast enough, this detracts from the usability), functionality (often all kinds of functionality are added to a system in order to make it more user-friendly) and continuity (error-resistance).

While usability is largely a subjective concept, over the course of time a multitude of publications on this subject have appeared. The best-known person in this area is undoubtedly Jakob Nielsen [Nielsen, 1999]. In addition, the World Wide Web Consortium has set up guidelines for the accessibility of websites so that they are also suitable for visually impaired people [www.w3c.org].

Importance of usability testing

The importance of usability has increased markedly with the rise in the digitization and computerization of society. Via the Internet, organizations have acquired new communication channels to their customers and the market, with new kinds of services (online auctions, instant price comparison). The website has become the company's shop window and business card. Usability increases in importance when the user can purchase the same service or product for the same price, either from a competing website or through a traditional communication channel, such as a shop or telephone helpdesk.

Example

Competing with traditional communication channels
The government has a monopoly on the supply of certain services or information. With web applications, substantial cost savings could be realized if enough citizens make use of these, rather than using the telephone, sending in forms or going to the town hall. However, if people prefer not to use the website, they will continue to use the traditional channels.

Other consequences of inadequate usability are that the users:

- Make more mistakes, resulting in all kinds of reworking operations
- Work less efficiently owing to confusion and more keyboard operation
- Do not know what they have to do and so make frequent calls to the helpdesk
- Require long periods of learning.

Example

Usability of the ATM

The early ATMs in the Netherlands involved the following sequence of operations:

1. Insert PIN card in machine
2. Enter PIN code
3. Enter cash sum
4. Receive cash
5. Remove PIN card

Your aim as a user of an ATM, i.e. to withdraw cash, is achieved at step 4. Users regularly forgot to remove their cards. This has been adjusted, by switching the last two steps. Now the card is returned first, and only then is the cash delivered. In other countries, such as the US, this adjustment has not yet been entirely implemented, as one of the writers found to his dismay …

While the usability of websites has greatly improved over recent years, this remains a risk factor for a successful site. The rise of electronic agendas and smart phones, too, is giving usability problems with websites for mobile use. But usability problems do not only occur in relation to websites or custom applications – they also affect, for example, embedded software and standard software packages. In the latter case, however, the possibilities for improving the usability are often limited.

Test design techniques

A number of techniques are available for the testing of usability. Worth noting here is that usability problems found at an advanced stage (such as the acceptance test) are often far-reaching and difficult to solve, for example because the application navigation or all the screen controls need to be changed. Usability and the testing of it should therefore be taken into consideration from the beginning of the design stage, when it is still possible to make relatively inexpensive adjustments. Possible test objects are, for example, apart from the working system, prototypes and screen designs. A few of the most important usability techniques are mentioned below. Roughly, they have the following characteristics:

- Moment of applicability
  Can the technique already be used for screen design; is a working system required or is the technique intended for a system that is already in production
- Testers
  Who evaluates the usability? This may be usability experts and/or the actual users.

Heuristic evaluation

Heuristic evaluation is one of the best-known ways of testing usability. During a heuristic evaluation, a systematic examination is carried out of the usability of the design of the user interface. The ultimate aim of heuristic evaluation is to discover problems in the design of the user interface. By finding such problems at the design stage, it is possible to solve them in time. During the process of heuristic evaluation, a group of 3-5 experts (evaluators) give their opinion on the user interface in accordance with a number of usability principles (also known as the "heuristics").

In more detail

Nielsen distinguishes 10 heuristics; see [Nielsen, 2006]:

- Visibility of the system status
- Match between the system and the real world
- User control and freedom
- Consistency and standards
- Error prevention
- Recognition rather than recall
- Flexibility and efficiency of use
- Aesthetic and minimalist design
- Help for users to recognize, diagnose and recover from errors
- Help and documentation

Usability test

In a controlled environment, a number of observers watch the way in which one or more users use the system. Besides usability experts, it is advisable to invite a number of designers for this. A few tasks are selected for the user to perform that are characteristic of the application.

In more detail

A task description typically consists of:

1. A sketch of the starting point, consisting of a description of the role that the subject assumes and their background, e.g. an inexperienced user or an experienced administrator
2. One or more tasks, e.g. check the status of the last order, compare the prices between two suppliers and order an item from the cheaper of the two. The task should indicate *what* has to happen, but not *how* the user should do it.

The subjects should read the role description and prepare themselves to carry out the tasks from that background.

During the execution of the tasks, the idea is that the subject continually thinks aloud and says what he or she is doing. For example, a reaction can be "I'm now going to the menu and opening the option of 'Information on company X', to see if I can find the route map there. Oh no, it's not here… (Etc.)".
The onlookers observe the behaviour of the user and take notes. In a so-called usability lab, the observers remain behind a one-way mirror and everything is recorded on video (both the images of the user and the images and operations on the computer). Another technique, such as eye tracking (the registering of eye movements on the screen) and other physiological measurements (heartbeat, perspiration) are possible here. Because of the infrastructure and equipment used, a usability lab is generally (very) expensive. A cheaper, but less effective, alternative to a usability lab is to have the observer sit with the user and, for example, just use a video camera or use a tool to register the user's actions on the system.
The observer(s) then assess the usability of the system on the basis of e.g. the number of mistakes made, the time taken to complete a task and the navigation path followed. They also use the participants' remarks during the test in their assessment of the usability.

Questionnaires

Another means of evaluation is to request the users' opinion of the system using questionnaires.
While they are also applicable to prototypes or even screen designs, questionnaires are mainly used when the system is ready, or even already in production. When the

participants have completed enough questionnaires, an evaluation of the results follows. While it is a relatively cheap method of testing usability, the disadvantage is that the result will not deliver a particularly detailed impression of what is right and wrong in a system. SUMI (Software Usability Measurement Inventory, http://sumi.ucc.ie) and WAMMI (Website Analysis and Measurement Inventory, www.wammi.com) are methods that are based on the use of questionnaires

Checklists, interviews

Cheap usability test techniques are the use of usability checklists during other (usually functional) tests, or interviewing the testers and users after working with the system concerning their experience of it.

Tools

Finally, tools are available, especially for web applications, that can carry out all kinds of checks. Examples of these checks are:

- Are the graphics and animations provided with an alternative (a text box) for supplying the same information in the event that the graphics, animations, etc. are not working? This can be the case if you use a different browser, don't have a video card or are visually handicapped
- Is the size of the graphics too big, making the site slow?
- Does every page contain a link for returning to the previous page and/or a link for continuing on to the next page?
- Are the text boxes perhaps too long in a scrolling field?
- Are all the links (still) valid?

4.5 Test environments

4.5.1 Introduction

A fitting test environment is required for testing a test object (running software). Setting up and maintaining the test environment represents an expertise of which testers generally have no knowledge. This is why a separate department – outside the project – is generally responsible for setting up and maintaining the test environment. Testers are, however, heavily dependent on the test environment – no test can be executed without a test environment.

This section discusses in greater detail what a test environment is and what its setup and maintenance look like. The section "Test environments explained" defines what a test environment is, after which section "Setting up test environments" describes the setup requirements for test environments. It also discusses the factors that determine the setup. The next section ("Problems in test environments") describes typical problems relating to test environments, followed by a solution to prevent these problems: the DTAP model in section "DTAP model".

Definition
A test environment is a composition of parts, such as hardware and software, connections, environment data, maintenance tools and management processes in which a test is carried out.

Hardware refers to all the tangible parts of a computer (screen, hard disk, network card, etc.). Test environment software refers to all the programs that should be present on the available hardware in order to run the software under test, such as operating programs, DBMS, network and other support programs. Connections are everything that is required to allow the test object to communicate with other systems. The environment data is the set

of data that the test environment requires to be able to work with these (user profiles, network addresses, root tables, etc.). Maintenance tools are tools that are required specifically to keep the test environment operational, and management processes are all the activities that are carried out around the setup and maintenance of a test environment.

The setup and composition of a test environment depend on the aim of the test. The success of a test environment depends on the degree to which it can be determined to what extent the test object meets the requirements. Every test may have a different aim, which is why every test can use a different test environment. A unit test, for instance, requires a completely different configuration of the test environment than a production acceptance test.

Sometimes a test environment has a limited size (e.g. one single PC when testing a small accounting package), while sometimes it involves a huge collection of hardware and software, interfaces and procedures, set up in many different sites (e.g. for testing the reservation system of an airline company). In addition to the test level and test type, other aspects - like the maintenance standards, the type of application, the organization structure and, not least, the available budgets - play an important part.

Test environments represent a critical success factor for virtually every automation project. There are various reasons for this. For instance, in a production environment the maintenance processes have been established for a long time and are still being improved. This does not apply to a test environment. Processes are not yet or partly established, and this may often vary per department and platform. The complexity increases further if the test environment also uses new technologies that have not yet been taken into production and with which the organization therefore has less experience.

Another development in recent years is that applications use an increasing number of different types of hardware and software. When setting up a test environment for this type of applications, this is translated to a chain of different hardware and software configurations with mutual interfaces. The metaphor 'the chain is as strong as its weakest link' then holds true. If one configuration or interface in the chain fails, the entire chain is useless and complete testing is impossible.

Furthermore, a problem or bottleneck in a test environment is not always quickly solved by an administrator. After all, production always has the priority. This is neglecting the fact that delays in the test process result in delays in commencement of production. Such delays can have the same (or worse) consequences as defects that occur in production.

4.5.2 Setting up test environments

4.5.2.1 Setup requirements

The degree to which it can be established in how far the test object complies with the requirements determines whether a test environment is successful. The setup and composition of a test environment therefore depend on the aim of the test. However, a series of generic requirements with which a test environment must comply to guarantee reliable test execution can be formulated.

Representative

The test environment must have the properties (as much as possible) that are required for the planned test. This does not mean that the entire test environment must always equal the production environment. For instance, for a functional test of an interface between two applications you do not need a complete environment that matches the future production environment.

Example

For the development of an application intended for eventual use on a UNIX platform, a Windows-based platform was used as the test environment for the system test. The assumption was that the functionality would not be affected by the platform difference. A UNIX-based test environment was used for the UAT and PAT.

Manageable

A manageable environment is required to test the test object under the same conditions every time. It must be clear at all times which version is installed in a test environment. This applies not only to the test object, but also to all of the software (i.e. the operating system, database management system, network protocols, etc). Changes in the components of the test environment (hardware and software, test object, procedures, etc) cannot be implemented unless with permission from the environment's owner (in projects, often the test management).

Flexible

A test environment must be easy to adapt. This may conflict with the previous requirement. Which of the two requirements (manageable or flexible) takes precedence, depends on the aim of the test and the phase of the test process. For instance, adjustments may be necessary when analyzing defects or implementing a new version of the software. It may also be necessary to create or eliminate specific connections with other systems. If this is done in a test environment of one project, which has no impact on anybody else, flexibility wins. In case of a shared environment (e.g. an end-to-end test environment), manageability is preferred. Other examples of possible changes are the system date and time, currency, calculation units and regional settings. Adjusting the system date and time may be necessary to make time jumps during testing. This is also called time travelling, making it possible for the system to be moved to the past or the future. It can be used, for instance, to run a system cycle of one year in just half a day. Changing regional settings is important when testing software that will be used in several countries.

Continuous

If there are disturbing situations in the test environment, one must try to continue testing as much as possible. The consequences of a failure must therefore be limited to a minimum. An important mitigating measure is making regular backups so that they can be restored if necessary. Furthermore, these secured initial situations can be used time and again for the test or to investigate a specific defect. Another mitigating measure is to create a fallback option for the test environment. The fallback option may consist of a second logical environment in addition to the existing test environment. The risk is that, if problems occur in the hardware, they affect both environments. Another option is therefore to set up a second physical environment. To limit the costs to some extent, the organization may decide to combine the second environment with the fallback facility for the production environment.

Example

When adapting an application that was used for annual contract renewals, it was necessary to perform tests on several dates and times (time travel). As such, easy modification of the system date was a requirement for the test environment. Furthermore it was necessary, due to the time travel, to create regular backups and restore them later. Not a complex combination of operations, but it did put a lot of work pressure on the administrators of the test environment. It was therefore decided to develop a menu screen containing the various operations and make it available to the testers. This relieved the administrators and allowed the testers to have better grip of their environment.

4.5.2.2 Factors determining the setup

Translating these requirements to the actual setup of a test environment varies for each test. For instance, the test environment for testing the screens in the system test may be different from that for testing security during the acceptance test. A large number of factors play a part in setting up the test environment. You will find a list of determining factors, with a summary explanation, below.

- The test level for which the environment is intended - unit, system or acceptance test or possibly a combined test.
- The test type for which the environment is intended - performance, usability, security or regression test?
- Requirements made by the external organizations for the environment, e.g. supervisors or (local or central) authorities.
- Requirements made for the test data to be used. Are they small or big volumes? What is the refresh rate?
- Existing test environments in the organization, if any. Can they be used? How can individual requirements be implemented?
- Is there a budget for setting up test environments and which options are available?
- Does the organization have standards for setting up test environments?
- The hardware and software architecture. Which development or production platform is being used? What are the options and which limitations exist, if any?
- The manner in which system development is organized. The methods, techniques and phasing used for system development have an impact on the test environments in terms of procedures.
- The type of system. Clearly the test environment has a strong relationship with the nature of the test object, e.g. batch, online, mainframe, PC application, custom or package.
- The level of distributed processing. What extent of data communication exists? And in what form? Is the network or network programming part of the test object? Are decentralized test sites used? Are there any interfaces with external organizations?
- Scope of the test. Should manual processes in e.g. input and output processing be tested as well?
- The test environments of the programmer and tester must not be too distant in terms of geography. While communication resources like telephone and e-mail may respond to part of the communication requirement, frequent consultation between the various stakeholders will be necessary. An optimal location choice can save a lot of time and money.
- Sometimes the use of test tools makes demands on the test environment in relation to e.g. security, data storage and communication resources.

Tip

The cube notation for test environments
A lot of characteristics must be recorded for test environments. Characteristics that are determinants for the identification of an environment, but also those about which an agreement has to be reached with other parties. The registration method for these characteristics partly determines the success of the various arrangements. When multiple test environments are involved, the clear and structured recording of the characteristics may be problematic. One way to do this is to work with the so-called cube notation. A number of characteristics are placed in each visible plane of the drawn cube. An example is shown in figure 57 "Cube notation of the various characteristics of a test environment".

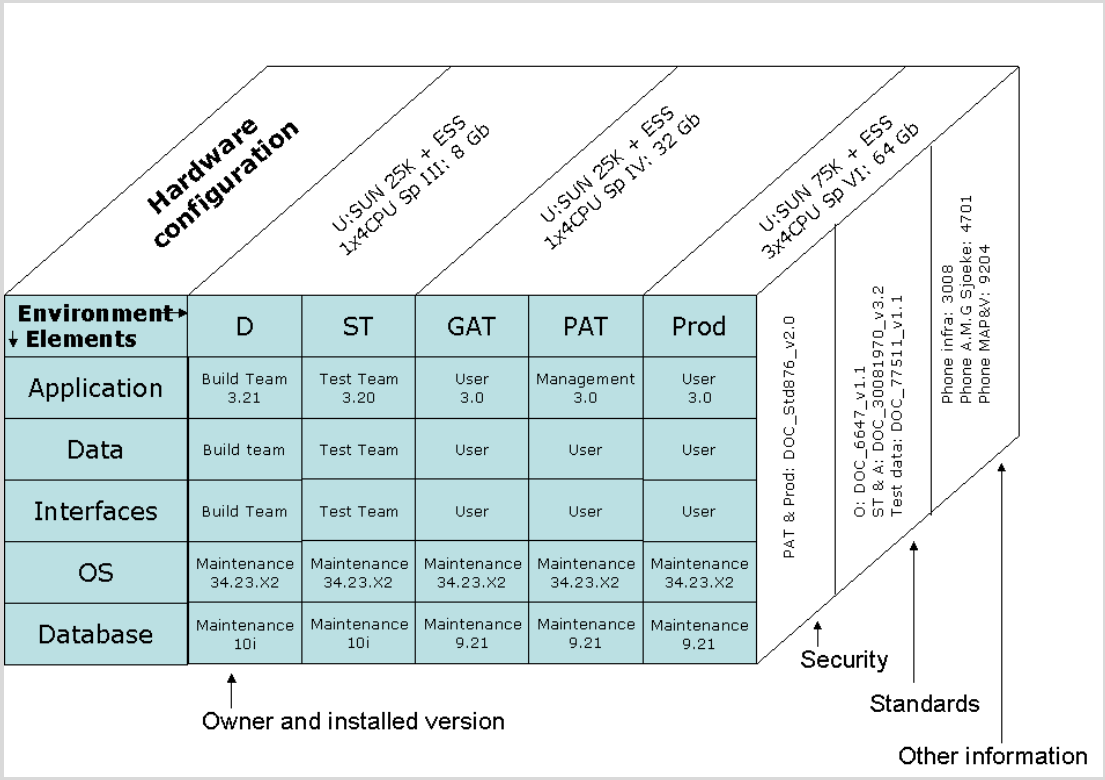


Figure 57. Cube notation of the various characteristics of a test environment.

This makes everything clear at a glance. We recommend hanging this plate in the common test or project space so that everyone can see the applicable arrangements at any time.

4.5.3 Problems in test environments

In automation projects, it often happens that many different environments are being used. An organization may have one or more development environments, one or more test environments, a production environment with a fallback environment and sometimes also several maintenance environments. In this situation, the following problems might emerge:

- Returning defects. A defect detected in version X is solved in version X+1 but suddenly reoccurs in version X+2.

- No guarantee that it still works. The development team cannot guarantee that everything still works despite the fact that the release covers only a limited number of defects.
- Unannounced new features. When testing a new version, it is found that specific features (new functionality, specific technical aspects) have already been realized while the testers are not aware of them.
- No connection between defect and environments. A defect detected in environment X does not occur in environment Y while they seem to be the same environments. E.g. a defect does present itself in the acceptance test environment, but not in the system test environment.
- Defects cannot be investigated. A defect cannot be investigated anymore because a user other than the tester has modified the test environment.

There are two solutions to prevent these problems. In the first place, the environments must be separated according to the DTAP model. That model and how it can be used is explained in the next section ("DTAP model"). In the second place, formal processes must manage the setting up and maintenance of the environments. The latter solution is not a part of this Workbook.

4.5.4 DTAP model

DTAP

DTAP stands for **D**evelopment, **T**est, **A**cceptance and **P**roduction. The basic principle of the model is that every user of the infrastructure wants to do his or her job undisturbed, without being hindered by anyone else. For instance, the end user does not want to be bothered by the tester, who in turn wants to be left alone by the programmer. This is why a separate type of environment is defined for each of these parties. The 4 environment types are analogous to the 4 stages software goes through: the software is developed (development), tested (test), accepted (acceptance) and used (production).

While the DTAP model may initially look like a technical solution, it is not. The model does not prescribe that there are 4 environments, but simply that there are 4 environment types. Each of these 4 types has its own characteristics. As such, the DTAP model makes allowance for the use of 7 environments, for instance, in a project (see figure 58 "Different environments in a development project according to the DTAP model"). There might be two development environments (local and centralized), one test environment, two acceptance environments (user acceptance test and production acceptance test environment), and two production environments (production and shadow).

Environment type acc. to DTAP	*Development*		*Test*	*Acceptance*		*Production*	
Environment in development project	Local development environment	Central development environment	ST environment	UAT environment	PAT environment	Production environment	Shadow environment

Figure 58. Different environments in a development project according to the DTAP model.

Owners and administrators of the environment types

Test activities can be executed in every environment type of the DTAP model. Since every environment has an owner, administrator (manager) and its own group of users, the

various activities have their own characteristics (see figure 59). For instance, the test environment type is managed differently than the production environment type. In the DTAP model, it is important to distinguish which parties are the owners or administrator of each type of environment. The owner is the party who determines which users are allowed and what the administrators need to do. In the DTAP model, the aim for which the environment is used determines the owner. Sometimes the owner is also the economic owner, but not necessarily.

In the development environment type, it is generally clear who the owner and administrator are. Both roles are fulfilled by the programmers. They acquire and maintain the environment. It is equally clear for the production environment type. The user organization is the owner, and often the maintenance is handled by a special maintenance organization (on behalf of the user organization).

For the test and acceptance environment types it is often a bit more complicated because multiple parties are involved. The user organization is the owner of the acceptance environment and the testers are the owners of the test environment. But the environments can be maintained by several parties. It may have been acquired by either the project or the maintenance department. In the latter case, the maintenance may be handled by the maintenance department or the testers themselves. The maintenance can even be in the hands of the developers.

Environment type	**Owner**	**Administrator**
Development	Developers	Developers
Test	Testers	Developers/ Testers/ Maintenance organization*
Acceptance	User organization	Developers/ Testers/ Maintenance organization*
Production	User organization	Maintenance organization

* = different possible options

Figure 59. The possible owners and administrators of the 4 environment types.

Test types and the 4 environment types

The DTAP model does not impose a consistent link of a test type to one environment type. This is to prevent negative consequences. Because of the consecutiveness of the test process in the test environments defects may be discovered too late. This can be prevented by executing a test type in more than one environment type. Clearly, the delivery of the testable parts of the test object must be related to the test type (and the associated environment). In this construction, the user may execute some tests in the development environment.

The realization of this model is a challenge for the test management and stakeholders. The owner of the environment must accept that his environment may be used for any test type. Different user groups can use the environment. The time gain that can be achieved thanks to parallelism of the tests and reduction of the repair costs due to earlier detection of defects are more than worth the effort. It is therefore especially important that the test environment fit the test type, in the DTAP model this is a perfect fit.

Tests in the development environment type

The unit test is executed in the same environment type in which the software and other system components are developed: the development environment. Setting up this environment and the related test activities are executed as part of the development

process. When a part of the environment must be used for a test, the developer himself is usually the party arranging this. Often the development platform contains standard facilities for testing, such as files, test tools and procedures for e.g. version management, transfer, defect administration and defect repair. These facilities offer the developers adequate options to manage their test process correctly. If there are no specific requirements for the unit tests and the above standard facilities are available, the tests can be executed correctly. An important aspect that programmers must deal with is the manageability of their environment. In practice, a programmer often has five or more versions of his software under management. Maintaining the relationship between the test cases, test results and the test object requires a lot of attention in this case.

Tests in the test environment type

The test environment type is created to test (parts of) the entire system for both technical and functional aspects. This test must be executed in a manageable environment. Manageable means that resources are available to transfer and manage, among other things, the software, documentation, test files and testware. The tester must be able to control the transfer of new or changed software. The tests must be reproducible. It must be possible to execute the individual tests of one (sub-)system separately from the tests of other (sub-)systems. The simultaneous use of the same test data in particular may cause a lot of trouble (see section 4.3 "Defining central starting point(s)"). In this environment type, tools can be used that provide the tester with insight at a technical level into various events. Examples are the use of SQL to look directly in the database, having direct access to the system's log files, and being able to start up and stop batches (see section 4.6 "Types of test tools").

Tests in the acceptance environment type

The acceptance environment type offers future users and managers the possibility to test the test object in an environment resembling the production environment as closely as possible. Usually the test in this environment type is split up into a user acceptance test and a production acceptance test. The UAT checks whether the test object provides the required functionality in relation to production facilities and procedures. The PAT checks whether the system complies with the management and production standards, in terms of both procedures and aspects like volume processing and performance. It is preferable to create a separate environment for the test types UAT and PAT, although it is naturally possible to execute them in the same environment.

In more detail

The PAT environment as a production environment

Organizations often feel that a test environment for the PAT is costly. Not surprising, because it is especially important for the PAT that the test environment is not only functional, but even more so technically equivalent to the production environment. Logically, this means that a PAT environment requires the same hardware as the production environment (types and quantities). As such, a PAT environment is a second production environment.

A solution is, in new development processes, to promote the PAT environment to production environment when the system is delivered. This means only one production environment is necessary. In maintenance projects, an option is to execute the PAT in a fallback environment, which is often a copy of the production environment. If there is no fallback environment, it can be decided to execute the PAT in the production environment at a moment when there are no users (e.g. at night or during the weekend). Clearly this last option involves some risk in terms of availability of the production systems – it is therefore recommended exclusively for relatively simple systems.

Tests in the production environment type

Testing in an environment that is used for production is not desirable, and sometimes even prohibited by regulatory bodies and other supervisors. In very exceptional situations, it is sometimes unavoidable to test in the production environment type. In these cases, the required test environment is so complex that it cannot be simulated or built. Example is a complex system chain (often across several organizations or even countries). In this type of cases, in-production testing is an option. But a lot of things have to be arranged for that purpose. For instance, the new version of the software must be accessible exclusively to the test team. Furthermore the execution of the test must not disturb the regular production process. Furthermore an (external) supervisor often checks the test execution because operations are executed (orders, payments, etc.) that are not formal.

4.6 Test tools

4.6.1 Introduction

The development in recent years that can be summarized as *'more for less, faster and better'* has an impact on all IT disciplines. With highly advanced development environments, developers can design and build complex programs relatively easily and quickly. The iterative development methods that are based on far-reaching interaction with the users ensure, among other things, that projects make interim deliveries faster. Such interim deliveries are then evaluated against the users' wishes and requirements and the defects are reworked in the software. This means that the software changes continuously and regression risks are always there. Moreover, development is based more and more often on reusing internal and external components that must be integrated into the existing IT architectures. This has reduced the time required to develop new systems, putting testing even more emphatically on the critical path in terms of development and maintenance. It even threatens to become an obstructing factor.

All these factors, taken together with the fact that system testing is already perceived to be a time-consuming and costly activity, make higher productivity of the tester and higher quality of the test a requirement. Test tools can be used as an instrument to achieve this.

Making test tools available to testers is often the responsibility of a separate department. One reason is the fact that setting up and maintaining test tools is a specific expertise. It is something of which testers generally have little knowledge. Another reason for making test tools the responsibility of a separate department is that big investments are often required to introduce tools in an organization. In addition to the high acquisition costs, investment is required in training the people and developing new procedures. In other words, it takes time to realize a return on investment, often longer than one single project.

This section discusses test tools and their use in greater detail. Section "Test tools explained" explains what a test tool is. Section "Types of test tools" then describes the various types of test tools. Section "Advantages of using test tools" discusses the advantages of using test tools. The subsequent sections describe how test tools can be implemented in test organizations on the basis of a tool policy. To this end, section "Implementing test tools with a tool policy" explains the concept of tool policy and describes the life cycle model. The three phases are then listed, i.e. Initiation (desired effects, commitment, preconditions), Implementation and Operation (use).

4.6.2 Test tools explained

Definition
A test tool is an automated instrument that supports one or more test activities, such as planning, control, specification and execution.

One of the conditions for the successful use of test tools is the existence of a structured test method of operation. In a properly controlled process, tools can certainly add a lot of value, but they are counter-productive in an inadequately controlled test process. In fact, test tools automate the test process, which requires a certain repeatability and standardization in the activities to be automated. An unstructured process cannot comply with these conditions. The deployment of test tools, however, can serve to leverage the implementation of a structured approach. However, the least that is required is structuring and automation combined.

In more detail

Terminology: tools, test tools and CAST tools
Tools used in a test process are referred to in different ways. For instance, some simply talk of *tools*, others of *test tools*, *CAST tools* (CAST stands for Computer Aided Software Testing), or *test automation*. It is not possible to make an unequivocal choice for the right terminology. There are parties that state that a tool is a test tool when it can be used exclusively to support a specific test activity. The counter-argument is that some test tools that serve to support test execution are sometimes used for other work. One example is a test tool that can be used to automate test execution. This tool works on the basis of automating operations and can also be used for data conversion. And that makes it a tool in a wider sense again. TMap uses the terms tools and test tools interchangeably.

Being able to use test tools is now assumed to be one of the tester's basic skills. However, being able to set up and manage test tools and the far-reaching automation of routine work (e.g. test execution) still requires specialist and in-depth knowledge of programming and tools. Not every tester has that knowledge. As a result, new types of specialism have emerged: test tool programmer, test tool expert, and test tool consultant.

In more detail

Price structure of test tools
There are all kinds of test tools, all with their own price structure. Commercial tools often have a licensing system where a one-off price is agreed based on the number of users of the tool. In addition to this one-off price an annual contract is signed, ensuring the organization that the tool's supplier will provide support and new updates and releases. Often, this is called a maintenance or service contract.

In addition there are test tools with price structure on the basis of the variants *shareware*, *freeware* and *open-source software*. The price structure for shareware is such that it can be distributed without or with few restrictions, but a fixed price having to be paid when used repeatedly. Freeware is software for which the author has issued a license for use and further distribution in unchanged form without requiring compensation. Open-source software goes one step further than freeware. In addition to the free distribution of the software, the author gives permission for modifying the software. The modified software can also be distributed freely.

Contrary to open-source software, freeware is protected fully by copyright. And contrary to open-source software, the source code of freeware is not usually made available. More and more (self-made) test tools are made available by the creators through the Internet on the basis of these variants.

4.6.3 Types of test tools

Test tools provide support in the execution of certain activities in the various TMap phases. There are different types of test tools, which can be classified in four groups:

1. Tools for planning and controlling the test
2. Tools for designing the test
3. Tools for executing the test
4. Tools for shaping the test environment.

4.6.3.1 Tools for planning and controlling the test

Like a business process can be supported by automated resources, a test process can be supported by automated instruments. These are test tools that support activities in relation

to planning and controlling the test, like creating the planning, monitoring progress, and registering defects. Because the tools focus on the process, in a technical sense they operate independently of the test object. The following tool types are in this group:

- Testware management tool
- Defect management tool
- Planning and progress monitoring tool
- Workflow tool.

In more detail

Test management tool not a separate tool type
Test management tools are not defined as a separate tool type in TMap. The reason is that it offers an integrated set of functionalities in the field of various tool types. For instance, a test management tool often supports testware management, defect management, and planning and progress monitoring. While the functionality for each field is not usually as comprehensive as in a specific tool type, the power of a test management tool lies in the integration of the various tools. Often the test management tools are also integrated with tools for automated test execution. The test management tool may also contain an automated workflow. This means that the tool supports the entire test process – from making the test plan to reporting on the results.

Testware management tool

All kinds of products are created in the course of the test process and together they form the testware. It is very important that the products are adequately managed during a test process. Testware management tools support the registration of the various versions of testware that are created in the test process and the possible relationships between the testware. For instance, it can be derived which test result belongs to which version of the test scripts, or which version of the test specification belongs to which version of the test basis. Furthermore, testware management enforces a certain level of structure and uniformity.

Defect management tool

These tools support the registration and handling of test defects found during a test process. The process of defect management is complex and voluminous. Sometimes the number of test defects, depending among other things on the size and quality of the test object, may amount to hundreds or thousands. Defects can also contain one or more annexes with screen prints or parts of the test basis to clarify the problem. Several parties, often in different locations, are involved in handling test defects. Sometimes the procedure to handle defects depends on the urgency of the defect. Tools are available to support these activities. In addition to the registration the lifecycle of a defect can be monitored and tracked. Some tools also enable the creation of management reports and metrics.

Planning and progress monitoring tool

A tool to support the process of planning and progress monitoring is indispensable in large-scale test processes. A planning must be calculated through and through in terms of activity time, start and end dates (if any), and allocated resources. Often, planning packages provide 'what if' analyses and are able to generate both strip planning and network planning units. These tools help with estimating the effect for the test. See www.tmap.net for an example. Progress monitoring must provide insight into the progress made, and reports on this must be generated. Furthermore it must provide insight into the required time and resources to complete the test process. An important aspect in the selection of tools for planning and progress monitoring is the possibility of creating management information, e.g. overviews of resources and costs.

Workflow tool

The TMap test process has various phases with activities and sub activities. Some of these are interdependent: the output of an activity is the input for another activity, resulting in multiple chains of activities (workflow). The activities in a chain are executed by one or more persons in the test team. In the case of large test teams, managing the entire process with the various activity chains is a complex task. A workflow tool can provide support. The workflow tool knows the activities to be executed and ensures that the work is routed to the relevant persons. With the tool, the test manager has continuous insight into the status of the activities to be executed, and is aware of the total work stock. The tool generates an alert when plans are exceeded or work stocks become unusually high so that the test manager can intervene.

4.6.3.2 Tools for designing the test

Tools that support the specification of test cases or generate them fully automatically belong in this group. This group also contains the test tools to create, set up and maintain the test data. Tools that support the creation of test cases usually do this on the basis of a coverage type. When the test basis is described in a formal notation, the test tools can generate test cases automatically. In many cases, these test cases require further processing. The tool provides support in this. The following tool types are in this group:

- Test design tool
- Model-based testing tool.

Test design tool

These tools provide support when test design techniques are used during the specification of test cases. In particular when various possible combinations of input are used during testing, these tools quickly add value.

Model-based testing tool

These tools offer support in the approach of Model-Based Testing. This is an approach in which test cases are designed on the basis of a model of the test object (figure 60 "Model-based testing"). These test cases are then used for automated execution on the test object. One of the challenges in this approach is the creation of a formal model in which the operation of (part of) the application is shown. Creating this model is work for humans. When the model is complete, it can be read by a tool that handles the creation and execution of test cases. This method is particularly valuable for (a combination of) complex systems that have an unlimited number of possibilities. For more information on Model-Based Testing, go to www.model-based-testing.org.

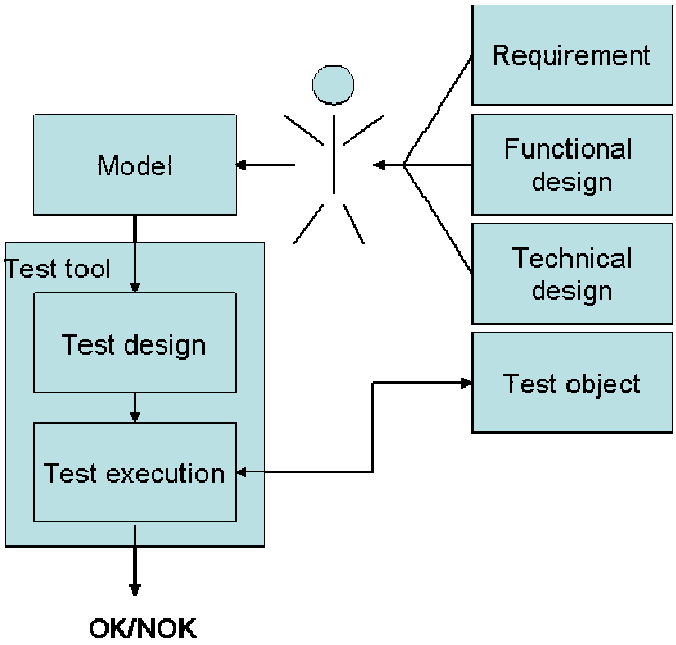


Figure 60. Model-based testing.

In more detail

Word-processing and spreadsheet programs viewed as test tools
It is sometimes said that the test tools most often used by a tester are word-processing and spreadsheet programs. At first sight this might seem a funny statement. But when looking beyond the standard functionality of these tools, there might be some truth to it. These tools can support a tester's work and in some cases even automate it. By simply copying and pasting pieces of text, reuse in the creation of test scripts is simplified. The use of a spreadsheet for the notation of the logical and physical test cases (in the different cells) imposes a standard work method, which benefits interpretation by the various testers. Furthermore most word-processing and spreadsheet programs contain so-called 'macro' functionalities to automate operations. In some cases, links can even be made to external programs. This makes it possible to automate an activity like test execution (in a very light form) with a word-processing or spreadsheet program.

Example

A batch system uses text files as input. The text files contain lines with various data. Each line consists of 10 or more data elements separated by a comma. The text files are used to test the batch system. Thus, testers must deliver their test cases in the format of the text files. Reading and understanding the content of a text file is difficult. The meaning of a data element depends on its position and value. Creating text files for testing is therefore very complex. It was decided to create the text files in a spreadsheet program. By linking each column to a position (and the meaning) in the text file, the testers can build the text file quite easily. A text file is then created for the batch system with a click of the button based on the various cells. See figure 61 "Use of a spreadsheet to create a text file" for a schematic representation.

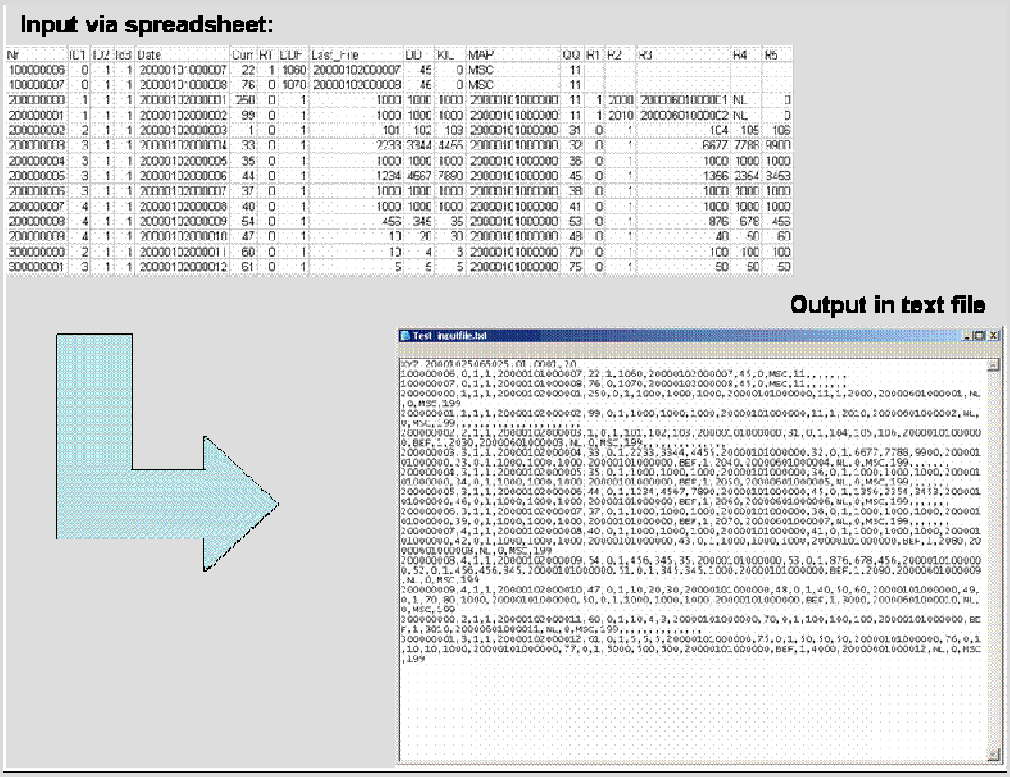


Figure 61. Use of a spreadsheet to create a text file.

4.6.3.3 Tools for executing the test

These test tools are deployed on the critical path of testing: executing test scripts. Because the tools focus on the product, they must, technically speaking, cooperate with the test object and the associated hardware and software combination. The deployment of this type of test tools is beneficial when the test work requires great accuracy and is relatively routine. Examples are the frequently repeated execution of the same test and comparing sizeable overviews with the aim of determining whether they are both the same. Also activities requiring a lot of technical knowledge (e.g. security testing) or many testers (e.g. testing with load profiles) can be executed by these test tools.

The following tool types are in this group:

- Automated test execution tool
- Performance, load and stress test tool

- Monitoring tool
- Code coverage tool
- Comparator
- Database manipulation tool.

Automated test execution tool

As the repeated testing of unchanged functionality (regression testing) is the most sizeable and time-consuming part of the test, tools for automated test execution are attractive to many organizations. Regression testing starts as early as when a system is being built and takes up an increasing part of test work during the life cycle of the system (see figure 62 "Increasing share of regression testing during the lifecycle"). The automated execution of such regression tests can save time. This is attractive not only to the tester, who is relieved of repetitive and therefore boring daily activities, but also to the calculating test manager who can save tens of percents.

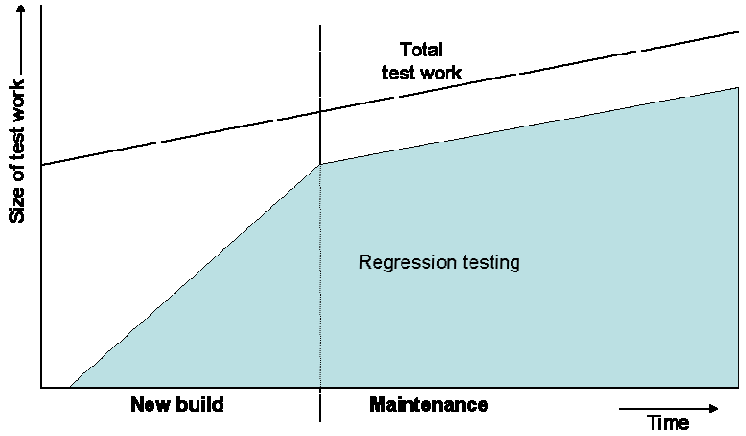


Figure 62. Increasing share of regression testing during the lifecycle.

There are two variants of this test tool type:

- Tools that automate test execution via the user interface (GUI) of the application to be tested. These are also called record & playback tools. A record & playback tool records the test input (data and actions) and the expected result in a script. The tool can play back the script at a later time, so that the test can be repeated easily (please note that the term 'script' in this context should not be confused with the manual test scripts that are part of the test specifications).
- Tools that automate test execution via a program interface. Examples of a program interface are Application Programming Interface (API) or messages in XML format. Often this tool type offers the possibility of mutating stored input data and provides support when generating test input. Generally speaking, these tools are combined with comparison tools to enable analysis of the test results.

The great advantage of automated test execution tools is that a test can be repeated by automation at a later stage. This advantage is nullified if the test object is changed in such a way that the automated script blocks during playback. Maintenance to the automated scripts is necessary to use the tool efficiently. Such maintenance should not cost more than the benefit yielded by automated test execution. Changes in the test object must result in a limited number of changes in the automated scripts. This is often the case in regression testing, so that this tool type is extremely suitable for this test type.

The combination of tool, framework, test cases, automated test scripts, and recorded results is called a test suite. The framework in a test suite is a library of reusable automated

scripts. Each script is in fact a small program. Use of the basic principles of modular programming increases the maintainability of the scripts: each group of successive actions that must be carried out repeatedly (for example moving to a certain screen in the application) is best stored as a separate module. If something changes in the group of activities (for example because of a different menu setup), then only one module will need to be adapted. Modules exist at different levels of abstraction, varying from activating or checking a specific object of the system to be tested, to carrying out a business process.

Having such an architecture makes it possible for new test suites (for new systems) to be created in a short period of time, because many of the necessary building blocks (modules) are already present in the library. To construct a test suite in such a modular fashion, expertise in the fields of testing and software development is required. The required effort to adapt a test suite (and therefore also the framework) for a new release must not outweigh the benefits of the use of the test suite. The main quality requirements for a test suite are: maintainable, flexible, robust and reusable (see also [Fewster, 1991]).

Performance, load and stress test tool

Performance, load and stress test tools can load an information system by simulating (large numbers of) users. The purpose of this type of testing is to determine whether the system continues to function correctly and at the required speeds under the expected production load. To determine the possible causes of problems in the measured results, these tools are often used in combination with monitoring tools.

Monitoring tool

Monitoring tools are used in the test process to gain insight into aspects like memory use, CPU use, network load and performance. All kinds of data relating to resource use are measured and saved and presented by means of a report. Configuring such tools is often complex. However, often a maintenance department already has monitoring tools to monitor the operational production environment, perhaps these can be used in the test environment as well. In performance, load and stress test tools, monitoring functionality is often an integrated component.

Code coverage tool

Code coverage tools yield information on which parts of the program code were used during a test. As such they provide practical support to measure the effect of the test design techniques used. The measurements are made at the program or subsystem level. In this way, it is established whether each program statement is executed at least once during testing. The conclusions drawn must be investigated because:

- 100% coverage of the program statements does not guarantee by any means that no defects remain! Compare section 14.2.2 “Coverage, coverage type and coverage ratio”.
- A test designed to achieve 100% coverage of the functional specifications does not generally automatically achieve 100% statement coverage.

Comparator

A comparator compares data and reports the differences. The latter must then be analyzed manually to determine whether the differences coincide with expectations. These tools are used to e.g.:

- Compare test output against the test output of the previous test
- Compare a data collection before and after one or more test actions
- Compare the results of shadow production against the results of production.

Such tools are often an integrated part of record & playback tools. As an alternative, the simple file compare functionality[7] or the revision functionality of a word processor can be used.

Data base manipulation tool

Directly viewing and manipulating data in a database represent a powerful instrument for testers. It enables them to execute checks to make sure whether a test was truly successful. This tool type is a vital part of the standard equipment of a tester. In addition to retrieving data, the data can also be changed. This can be used to create start situations. The manipulation language on which such tools are based is often SQL[8].

4.6.3.4 Tools for shaping the test environment

In many cases, a production-like test environment is not available just like that. There are many possibilities to use tools to shape the test environment in the right way:

- Simulator
- Stubs and drivers
- Test data tool

Simulator

A simulator simulates the operation of the environment of the (part of the) test object to be tested. A simulator is used to test software for which it is too costly, dangerous or even impossible to test the actual environment, e.g. testing the operating software for an aircraft or nuclear reactor. The simulator communicates with the test object as if it were the actual environment. It supplies input to the test object and receives its output. Simulators are generally not standard and must be developed in parallel with the development of the test object. The simulator in turn must also be tested.

Stubs and drivers

A system is generally tested in parts. A part may be a module or component. To test a module that has relationships with modules not yet realized at an early stage, you need stubs and drivers that replace the missing modules. A stub is accessed from the module to be tested, a driver accesses the module to be tested (see figure 63 "Stubs and drivers in relation to module A and module B").

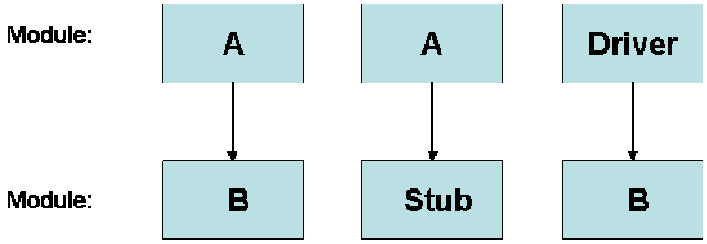


Figure 63. Stubs and drivers in relation to module A and module B

7 Often included by default in the operating system.

8 Structured Query Language.

Example

A reporting function that prints the payroll per employee is tested. In this function, the payroll calculating program (tested earlier) is accessed. The test aims to select all employees and print the payroll for every employee. However, preparing a test database with all of the required data for the various payroll calculations can be a huge task. A stub that returns a specific salary amount (e.g. based on the entered employee number) can significantly reduce the test effort. Naturally, the relationship between the real programs must always be tested once.

Test data tool

This tool helps build physical sets of test data. Using generators, random content can be created on the basis of a file and/or database specification. This makes it possible to create a sizeable set of test data relatively quickly, for instance for a real-life test. The 'rules' to generate test data must be pre-defined in the tool. Think, for instance, of defining collections with boundaries from which a selection can be made and relationships between various data types (consistency rules).

4.6.4 Implementing test tools with a tool policy

Tool policy

The activities in the test process supported by a test tool and how this will be set up depends on the tool policy pursued in the organization.

Definition

The tool policy describes how an organization handles the acquisition, implementation and use of test tools in the various situations.

The tool policy is part of the test policy. The tool policy describes in a uniform manner what the purpose of the implementation of test tools must be. The use of test tools is never an objective in and of itself. The test tool is just a means to realize a specific objective in terms of time, money and/or quality. This is called the test tool objective. The tool policy also describes the requirements, wishes and conditions (if any) defined for test tools. These can be based on requirements, wishes and conditions defined in the test policy.

Furthermore, the tool policy describes the approach to be followed for the acquisition, implementation and use of tools. As such, this part of the tool policy resembles a general plan of approach, with the difference that it represents the basis for a long-term investment. It has been written before: the deployment of tools usually only yields a return on investment in the long term, which is why it must be governed by a policy. A tool policy constitutes the basis on which the organization can base the use and implementation of tools (in the future). It is not a one-off document that is archived. It must be updated and adapted to new developments and insights continuously.

Example 1

A package supplier pursues a policy of acquiring building and test tools as much as possible from one single supplier. This is incorporated into the tool policy, which contains a list of preferred tool suppliers.

Example 2

The objective of an organization is to migrate all systems to a new hardware and software configuration within 3 years. The tool policy therefore specifies that new test tools can be purchased only if they will also operate in the new hardware and software configuration. The tool policy also states that the deployment of automated test execution must have a ROI within 2 years. The reason is that systems will change during the migration to the new hardware and software migration, meaning that the automated test execution will also change.

Example 3
An organization listed on the stock exchange must comply with legal requirements. These specify conditions with which the organization's systems must comply. These conditions can only be tested with specific test tools, which in turn must comply with these conditions. The list of test tools that comply is incorporated into the tool policy.

Example 4
A medium-sized organization has included a requirement in its tool policy that the organization does not need to have knowledge concerning the deployment of load, performance and stress test tools. This results in the fact that all performance tests are executed by an external supplier.

Example 5
The tool policy of a power supplier states that every project must use the standard available test management tool. Other tools must be open-source tools by preference. Commercial tools can be bought only with permission from the IT manager.

4.6.4.1 Initiation phase (desired effects, commitment, preconditions)

The first phase in the life cycle model is the Initiation phase. This phase contains activities that serve to obtain a univocal picture of the applicability of a test tool in a specific situation. An important condition for applicability is the information in the tool policy. Based on this, a well-considered decision concerning the deployment of and investment in test tools is made. The main activity in the Initiation phase is the execution of the *quick scan*. This provides information on the technical environment, the maturity of the test process, and the management's expectations concerning the deployment of test tools. Characteristics of the quick scan are its limited lead time and relatively low investment. Various other activities are possible in addition to the quick scan. Think of product presentations, a demo session, and visiting operational test tools in other organizations.

The quick scan

The quick scan is the instrument used to obtain specific information concerning the implementation of test tools. It has not yet been established whether tools will be used and which tools they should be. The aim of the quick scan is to collect and report information with a relatively low effort (from 2 to 15 days duration) about the possible applicability of a test tool in a specific situation. This results in a first (rough) version of the so-called business case for the implementation of tools.

An important source of information during the quick scan is the interviews. These are conducted with the main stakeholders in the test process. Examples are:

- Line manager (responsible for finance)
- Project manager
- Test manager
- Test consultant
- Application expert

- Developer
- Technical system administrator.

In addition to taking interviews, the quick scan also assesses various test products for their usability in a test tool. It is investigated in how far existing products, such as test cases or defect procedures, match the work method of test tools. Three aspects to determine the applicability of a test tool are taken into consideration. These are:

- Test tool objectives:
  The quick scan inventories to what extent the test tool objectives have been defined and are in line with the objectives as described in the tool policy. Often those involved only have expectations at that point, which may not prove realistic or translatable to concrete objectives. A precondition for the successful introduction of test tools is a client who is aware of the opportunities in the existing test process. These opportunities are the basis for concrete improvement goals. Based on these improvement goals, the objectives of the introduction of a test tool are compiled and rendered as concrete as possible. This makes it possible to render the achieved results measurable later.
- Infrastructure and test object:
  The infrastructure (test environments, workplaces and possibly other tools) and the test object play a vital part in determining the added value of a test tool. It must be investigated whether a tool matches the test object and the technical environment. This is a requirement in particular when the organization opts for automated test execution. Another important aspect is to investigate whether there are special test tools for the test object and, if so, what their possibilities are. This occurs often, especially when the test object is a standard package.
- Test method of operation:
  The implementation of tools as an efficiency measure adds value in particular when the processes are repeatable and predictable. In addition the process method must be supported by the tool – a test tool that serves to support defect management, for instance, must fit into the process of defect management. Location is a key aspect. When a test organization works in several physical locations, the tool will have to support this as well. In the example of the tool for defect management, it must be accessible from the various locations and all testers must work with the same database.

The results of the interviews and the assessments of the various test products are used to compile a first (rough) version of the business case. The most important aspects here are the expected investments and the expected benefits. This is a mix of tangibles and intangibles, which is why it is always very difficult to create a business case. Moreover, many things are still unknown after the quick scan. For instance, no specific test tool has yet been selected. A first version of the business case will therefore consist of the benefits expected by the stakeholders and an estimate of the costs that must be incurred to use the test products in a test tool. A business case can also consist of several scenarios elaborating the deployment of different tool types.

In more detail

Tangible and intangible benefits of tool deployment
Defining the business case for the deployment of test tools is always difficult. Partly because fixed and variable costs are involved, and partly because tangible and intangible benefits are involved. A reduction of the lead time is an example of a tangible benefit. But often there are also indirect benefits that do not have a direct bearing on money. For instance, the test organization's image will improve. It will radiate professionalism. Users and maintenance organizations like to see demonstrations of

automated testing. The test organization is more often asked to help with a variety of events. Employees will become more motivated. New career opportunities: technical specialization and working with modern tools.

A report is created on the basis of the results of the interviews, the assessment of the various test products, and the business case. In addition to the business case, the report contains a conclusion focusing on the possible deployment of test tools. In addition to this conclusion, it makes concrete recommendations for the follow-up process and which steps must be taken by which people.

4.6.4.2 Implementation phase

The second phase in the model is the Implementation phase. Based on a *plan of approach* that has to be created, all of the activities are executed and products are realized that are necessary to use a test tool in an organization. The aim of the Implementation phase is the implementation of a test tool, including the required *configuration*. Another part of this phase is elaborating the *preconditions* to enable use of the tool. Three sub-phases, associated with these three parts, can thus be distinguished:

1. Plan of approach
2. Setup preconditions
3. Test tool configuration.

The sub-phases are executed in parallel, making it possible to take account of findings (e.g. due to advancing insights) from one sub-phase in the execution of another sub-phase. Also this is time saving.

Sub-phase: Plan of approach

The quick scan provides information to create a first draft of the plan of approach. It describes the first setup of the preconditions. We recommend beginning with a test tool selection and the execution of the pilot. These are explicitly included as activities in the plan. At the end of the pilot, the plan of approach can be updated and concretized. The main aim of a plan of approach is the univocal definition of aspects like the objective, activities, planning and deliverables. Examples of subjects listed in the plan of approach are:

- Test tool objective
- Preconditions
- Pilot approach
- Configuration approach
- Activities
- Planning
- Products
- Organization.

Sub-phase: Setup preconditions

A number of preconditions must be met to enable the use of test tools. The main precondition is clearly the presence of a structured test process in which the use of tools may result in improvements. In addition to preconditions enabling deployment of the test tool, there are preconditions that must be met to enable use of the tool by the testers. The testers must be able to use the tool not just for current test work, but also for future test work. The way in which the preconditions are elaborated may depend on what is specified in the tool policy. Which preconditions must be met and how they must be set up depends on the specific situation. However, a number of generally applicable conditions can be identified:

- Test tool selection

- Pilot
- Business case
- Management commitment
- Maintenance in the line
- Trained testers
- Structured test process
- Communication.

These are explained in further detail below.

Test tool selection

There is a large variety of test tools that can be deployed in a test process. The specific environment and objectives determine which test tool(s) are most suitable in any situation. The strategy (for the future) of the test tool supplier is also gaining importance in the selection of a tool. Because an increasing number of test tools are integrated, selecting a product often also means selecting a supplier. If no test tool is available in the test, a *test tool selection* is done. Several approaches are available to this end that strongly resemble a regular package selection. Various tools are assessed on the basis of a pre-defined list of criteria. The criteria depend on the tool type for which the test tool selection is done. A list of example criteria can be found on www.tmap.net under "test tool selection criteria".

Pilot

The introduction of a test tool is not a standard process that can be done the same way in every situation. Every test has its own pitfalls. Often, many people have very high expectations from test tools. People are usually not aware that the deployment of tools requires an investment, the benefits of which do not usually become visible in the short term. Therefore one must proceed very carefully when implementing tools to avoid losing out to the difference between expectations and reality. By starting with a pilot project, insight into the added value of a test tool is provided in a relatively limited environment and in the relatively short term. The tool can be used on a small scale in a pilot, for example by part of the team or by testing a specific function. This makes it possible to evaluate the feasibility of the test tool aims. With a limited effort, insight thus is provided into whether the test tool is technically feasible, whether it matches the current test method, and the expected costs and benefits.

In more detail

The dip in the performance curve or why a pilot is necessary

What are the consequences for the employees of introducing test tools? This can be explained with figure 64 "Performance dip in the introduction of a new work method". The figure describes the situation in which an organization wants to improve its performance in a certain field. Current performance is at level M1. The organization wishes to perform at level M4. A new work method is introduced to achieve this.

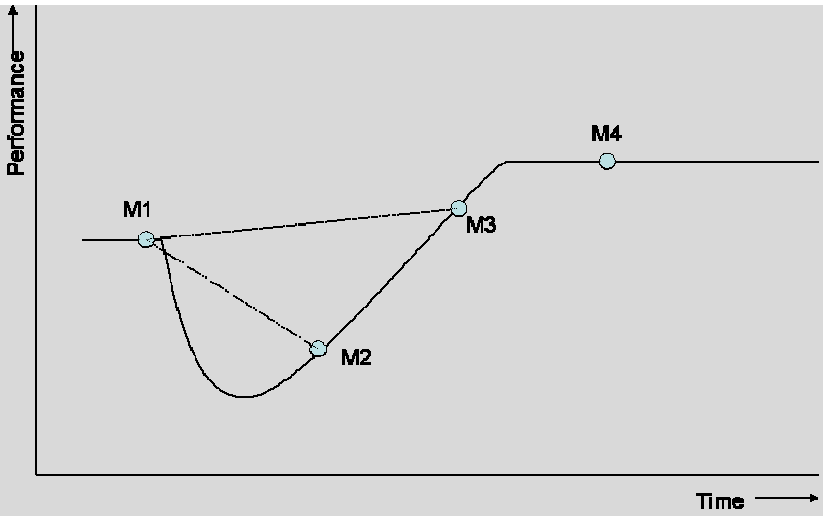


Figure 64. Performance dip in the introduction of a new work method.

The figure shows how the introduction of a new work method initially causes a dip in the performance curve. The path to move from performing at level M1 to performing at level M4 is not a straight upward line. A new work method must first be learned and, in most cases, adapted to the specific situation. If the stakeholders are not aware that the performance level will drop initially, there is a danger of measuring too early. The (lower) benefits of level 2 are then measured. It is concluded that this is not the right work method and another solution must be found. Often, the use of test tools is interrupted and the tool is shelved (also known as "shelf ware").

We recommend measuring the benefits of the new work method when the rising line compared to level M1 is started. In this example, this is point M3. It is an assessment that is difficult to make. When, in the introduction of a new work method, the organization also opts for a long chance process, the danger of overly early measurement and drawing the wrong conclusions is even greater. This is one of the reasons for using a short-term pilot. The dip will certainly show up during the pilot, but the dip in a "real production situation" will be smaller (and at least more easily predictable) based on the learning points and findings of the pilot.

Business case

The first version of the business case created in the Initiation phase is elaborated further. The figures in the business case can now be concrete. When specifying the costs, the fixed and variable costs must be taken into account. Fixed costs may be: hardware, licenses, installation, maintenance and training. Variable costs may be: test script creation, execution of test scripts, analysis of results, test script maintenance and training.

Management commitment

Even when the testing is sufficiently mature to use tools, it is not always certain that the desired benefits are realized. One of the main success factors for the deployment of test tools is the management's commitment. The management must be made aware that the use of the tool is an investment that usually yields an ROI in the longer term in terms of faster and/or better testing. If this awareness is inadequate, there is a great risk that the tool is taken out of production after the very first disappointment. This is even more true when the tool is deployed for the first time in a project with a fixed end date. If the project experiences time pressure, there is a great risk that the tool is taken out of production.

Maintenance in the line

An operational test tool may consist of a large number of items: modules in the test tool, framework, test data files, documents for use and maintenance, etc. All of these items must be maintained to enable reuse in the future. Test tool deployment only pays back over longer periods and therefore often across projects. By assigning the maintenance of tools to the line, knowledge retention is guaranteed.

Trained testers

The testers must be trained when the test team has not yet worked with a tool. Both the tool and working with it are new for the test team. The testers must acquire knowledge to ensure good use and maintenance. Training staff thus focuses on two aspects: gaining knowledge of the tool and of the use of the tool in the test process.

Structured test process

The deployment of test tools focuses on improving the test process in terms of money, time and quality. As such, it contributes to improving the efficiency of the test process. Before a test process can be made more efficient, it must be executed in a controlled manner. It may be necessary to define additional activities in the context of controlling the test process. For instance the use of test design techniques (see chapter 3 "Website"), which also increases the traceability of the test process. The measures to be implemented are highly situation-specific. We recommend using an improvement model (e.g. TPI) when implementing the improvement.

Communication

When the test team and the rest of the organization are not familiar with working with test tools, the aspect of communication requires extra attention. The stakeholders are informed of the plans in the field of test tooling as early on as possible. What are the plans, why are they executed, who is executing them, what are the planned results, and when will they be realized. We strongly recommend using the available communication resources for such communication. For instance a regular work meeting, a newsletter and the intranet. When these options are not available, information sessions should be organized.

Test tool configuration

In many cases, the test tools support a specific work method. Often this work method deviates from the situation in the organization that will use the tool. The tool must therefore be configured (see figure 65 "Configuration of the test tool"). Test tool configuration to ensure that it is in line with the organization is customization work. It involves activities like setting standard tables, defining a workflow, or programming a framework for automated test execution. The basis for the configuration is formed by the three aspects discussed in the Initiation phase. These are test tool objectives, infrastructure, and test object and test method of operation. A configuration plan is created on this basis. It describes concretely what and how the tool will be configured. This is vital to maintain the tool and its specific configuration in the future. The tool is then configured on the basis of the plan. We recommend doing this in collaboration with the future administrator of the tool, or asking him to do this. This ensures the first knowledge transfer. During the tool configuration, the configuration must also be tested. Any defects showing up in this test can be solved or incorporated in the configuration plan as known problems during completion. When a new version of the tool is developed, it can be examined whether these problems can be solved then.

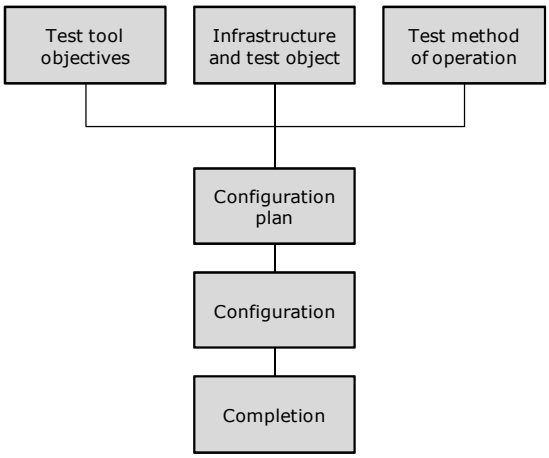


Figure 65. Configuration of the test tool.

4.6.4.3 Operation phase (use)

The third phase in the model is the Operation phase. This phase starts when the test tool is taken into production by the test team. To ensure that the test tool can continue to be used, maintenance must be executed. The use of the test tool will be part of the regular test process. This means that new activities must be executed by both the testers and the test manager. This also means that these people must be able to use and maintain the test tool the right way. The tool must have a place in the regular test process. When using the test tool, data must be collected on its functioning. Does its functionality fit in with the overall work method? If this is not the case, it must be examined whether this can be changed. The same applies to the evaluation of the test tool aims defined in the Initiation phase. It must be checked periodically whether the aims are still realized with the implementation of the tool.

One of the main principles in the use of test tools is the aspect of maintainability. The actual maintenance occurs in the Operation phase. When automated execution is used, issues like new releases, changes and incidents in the test object will have an impact on the test suite. But new releases of the test tool itself may also result in changes. These can be implemented, if necessary, after which the test tool is ready for use again.

Three types of maintenance can be distinguished:

- Technical maintenance
  The installation of the test tool (on the server or workplaces), implementation of new versions or patches, solving technical incidents, etc. Often the maintenance department that also handles the technical maintenance of other applications in an organization is responsible for this.
- Operational maintenance
  Enabling users to work with the test tool. This may involve issuing authorizations or configuring project-specific components (e.g. database). Often the maintenance department that also handles the technical maintenance of the tool may be responsible for this. Another option is to allocate the maintenance within the test project itself or to a permanent test organization.
- Functional maintenance
  Enabling users to work 'well' with the test tool. This means creating work instructions, manuals for the organization's own work method, procedures, templates, etc. It is important that functional maintenance does not maintain the functionality of the tool

itself – this is the supplier's responsibility. The test project itself or a permanent test organization may be responsible for this aspect.

The three maintenance types have separate responsibilities, but clearly they must collaborate as well. For instance, when a new version of the test tool becomes available, functional maintenance assesses its added value and impact. Functional maintenance then determines whether the new version must be introduced and when. Functional maintenance then directs technical maintenance to handle the implementation.

4.7 Defects management

4.7.1 Introduction

Many people see the finding of defects as the purpose of testing. While it should be clear that the purpose of testing is much more, i.e. the provision of information and advice concerning risks and quality, the fact remains that finding defects is one of the most important activities of testing.

A defect is also termed a 'fault'. Confusion sometimes arises concerning the various terms, such as errors, faults and failures. In this book, the following distinction is made:

- Error
  Human mistake; this action takes place prior to any faults and/or failures
- Fault
  Results from an error. Fault is the view from inside the system. Fault is the state where mistake or error exists. Developers will see the fault
- Failure
  When the system is performing differently from the required behaviour, from a viewpoint outside the system. Users will see the failure.

Within TMap, the following definition of defect is used:

Definition A defect (fault) is the result of an error residing in the code or document.

A defect can be found in several objects, like the test basis, the system that is being tested, the test infrastructure, etc.

Testers should realize that they are a) judging another's work, and b) that the final product is the result of cooperation between all parties. It is more considerate towards the other party to find a discrepancy between what the software does and what the tester expects, based on the available information, than if the tester immediately exclaims that he has caught the developer out in a mistake. The latter has a polarizing effect and quickly becomes a discussion on who has made the mistake, instead of a discussion on how best to solve the defect. In some cases, the testers employ terms such as "issues", "problems" or "findings" rather than defects. The tester should adopt as neutral an attitude as possible in connection with defects. Another good reason for adopting this attitude is that the cause of the defect often turns out not to lie with the developer, but with the tester himself. In a situation in which developers and testers stand opposite each other instead of side by side, a number of unjustified defect reports can destroy the testers' credibility entirely.

Administering and monitoring the defects also involves the solving of them. This is actually a project matter and not specifically a matter for the testers, although testers have the greatest involvement here. Good administration should be able to monitor the life cycle of

a defect and also deliver various overviews, which are used, among other things, to make well-founded decisions on quality. This management is sometimes assigned to a dedicated role: defects administrator.

From within the test process, the testers are the submitters of defects, and they check the solutions of these. The test manager communicates with the other parties concerning (the handling of) the defects. The choice may also be made to place this task within a separate role in the team: the intermediary. The purpose of this is to channel the defects and the associated solutions effectively. In this regard, the intermediary maintains external contacts at the level of staff doing the actual work. This person has an overview of all the defects and acts as a relay and inspection post for the defects on the one hand, and the solutions on the other. Advantages of this are that the quality of the defects and solutions is monitored better and that communication is streamlined.

There are great advantages to be gained in organizing one single defects administration and defects procedure for the entire project or the entire line department. All the parties involved – developers, users, testers, QA people, etc. can deposit both their defects and solutions here. Communication on the handling of the defects is thus considerably simplified. A central administration also offers extra possibilities of obtaining information. The authorizations are a point to note here; it should not be possible for unauthorized persons to be able to amend or close defects by this means.

4.7.2 Finding a defect

Defects may be found practically throughout the entire test process. The emphasis, however, is on the phases of Preparation, Specification and Execution. Since, in the Preparation and Specification phases, the test object is normally not yet used, in these phases the testers find defects in the test basis. During the Execution phase, the testers find differences between the actual and the expected operation of the test object. The cause of these defects, however, may still lie within the test basis.

The steps that the tester should perform when a defect is found are described below:

- Collect proof
- Reproduce the defect
- Check for your own mistakes
- Determine the suspected external cause
- Isolate the cause (optional)
- Generalize the defect
- Compare with other defects
- Write a defect report
- Have it reviewed.

The steps are in a general order of execution, but it is entirely possible to carry out certain steps in another sequence or in parallel. If, for example, the tester immediately sees that the defect was found previously in the same test, the rest of the steps can be skipped.

4.7.2.1 Collect proof

At a certain point, the test object produces a response other than the tester expects, or the tester finds that the test basis contains an ambiguity, inconsistency or omission: a defect. The first step is to establish proof of this anomaly. This can be done during the test execution, for example, by making a screen dump or a memory dump, printing the output, making a copy of the database content or taking notes.
The tester should also look at other places where the result of the anomaly could be visible. He could do this, for example, in the case of an unexpected result, by using an Edit

function to see how the data is stored in the database and a View function to see how it is shown to the user.
If the defect concerns a part of the test basis, other related parts of the test basis should be examined.

4.7.2.2 Reproduce the defect

When a defect is found during test execution, the next step is to see whether it can be reproduced by executing the test case once more. The tester is now on guard for deviant system behaviour. Besides, executing the test again helps with recognizing any test execution errors. If the defect is reproducible, the tester continues with the subsequent steps. If the defect is not reproducible and it is not suspected to be a test execution error, things become more difficult. The tester executes the test case again. He then indicates clearly in the defect report that the defect is not reproducible or that it occurs in 2 out of 3 cases. There is a real chance that the developers will spend little or no time on a non-reproducible defect. However, the point of submitting it as a defect is that this builds a history of non-reproducible defects. If a non-reproducible defect occurs often, it may be expected to occur regularly in production as well and so must be solved.

Example

During a system test, the system crashed in a non-reproducible way a couple of times a day. The test team reported this each time in a defect report, but the development team was under pressure of time, paid no attention to this defect, and dismissed it as an instability of the development package used. By reporting the large number of non-reproducible defects and indicating that a negative release recommendation would result, they were finally persuaded. Within a relatively short time, they found the cause (a programming mistake) and solved the problem.

In more detail

In some cases, such as with performance tests and testing of batch software, it costs a disproportionate amount of time to execute the test again. In those cases, the test to see whether the defect is reproducible is not repeated.

4.7.2.3 Check for your own mistakes

The tester looks for the possible cause of the defect, first searching for a possible internal cause. The defect may have been caused, for example, by an error in:

- The test specification or (central) starting point
- The test environment or test tools
- The test execution
- The assessment of the test results.

The tester should also allow for the fact that the test results may be distorted by the results of another test by a fellow tester.

If the cause is internal, the tester should solve this, or have it solved, for example by amending the test specification. Subsequently, the tester repeats the test case, whether in the same testing session or in the following one.

In more detail

Test environments and test tools usually come under the management of the testers. Defects in these that can be solved within the team belong to the internal defects, and those originating from outside the team are external defects.

4.7.2.4 Determine the suspected external cause

If the cause does not lie with the testing itself, the search has to widen externally. External causes may be, for example:

- Test basis
- Test object (software, but also documentation such as user manuals or AO procedures)
- Test environment and test tools.

The tester should discover the cause as far as possible, as this would help in determining who should solve the defect and later with discerning quality trends.

Because the tester compares his test case against the test object, there is the inclination in the event of an anomaly to point to the test object as the primary cause. However, the tester should look further: perhaps the cause lies with the test basis? Are there perhaps inconsistencies in the various forms of test basis?

As well as the formal test basis (such as e.g. the functional design or the requirements), the tester regularly uses other, less tangible forms of test basis. These may include the mutual consistency of the screens and user interface, the comparison with previous releases or competing products, or the expectations of the users. See also section 6.5 "Preparation Phase". In describing a defect, it is thus important to indicate which different form of test basis is used, and whether or not the test object corresponds with the formally described test basis, such as the requirements or the functional design. If the test object and the formal test basis correspond, the cause of the defect is an inconsistency between the informal and formal test basis and not the test object.

Example

During an exploratory test, the tester discovers that the position of the operating buttons vary on many screens. Further investigation shows that the cause lies with the screen designs and not with the programming. The tester submits the defect, citing the test basis as the cause.

An external defect is always managed formally. This may be in the form of the defect report and defects procedure described in the sections below. Where reviews are concerned, a less in-depth form may be chosen in which the defects are grouped into a review document and passed to the defect solver; see also section 4.12 "Evaluation Techniques".

4.7.2.5 Isolate the cause (optional)

While the suspected cause is often apparent, in the case of a defect in the test object or the test environment, it is often insufficiently clear to the defect solver. The tester therefore looks at surrounding test cases, both the ones that have been carried out successfully and the ones that have not. He also makes variations where necessary to the test case and executes it again, which often results in indicating a more exact cause or allows further specification of the circumstances in which the defect occurs. This step is optional, since it lies on the boundary of how far the tester should go in respect of development in seeking the cause of a defect. It is important to make agreements with the developer on this beforehand. This can avoid discussions on extra analysis work later on, when test execution is on the critical path of the project.

4.7.2.6 Generalize the defect

If the cause appears sufficiently clear, the tester considers whether there are any other places where the defect could occur. With test object defects, the tester may execute similar test cases in other places in the test object. This should be done in consultation with the other testers, to prevent these tests from disrupting those of his colleagues.
With test basis defects, too, the tester looks at similar places in the test basis ("In the functional design, the check for overlapping periods for function A has been wrongly specified. What is the situation as regards other functions that have this same check?").

Example

During a Friday-afternoon test, the parallel changing of the same item by two users in function X produced a defect. Further testing on other functions showed that the multi-user mechanism had been wrongly built in structurally.

The tester need not aim for completeness here, but should be able to provide an impression of the size and severity of the defect. If the defect is structural, it is up to the defect solver to solve it structurally. This step also has the purpose of building up as good a picture as possible of the damage that the defect could cause in production.

4.7.2.7 Compare with other defects

Before the tester writes the defect report, he looks to see whether the defect has been found previously. This may have been done in the same version of the test object by a fellow tester from within a different test. It is also possible for the defect to have been reported in an earlier release. The tester consults with the defects administration, his fellow testers, the test manager, defects administrator or the intermediary to find out.

There are a number of possibilities:

- The defect was found in the same part of the current release.
  The defect need not be submitted. The test case in the test execution report may refer to the already existing defect.
- A similar defect has already been found in another part of the current release.
  The defect should be submitted and should contain a reference to the other defect.
- The defect has already been found in the same part of the previous release.
  If the old defect was to have been solved for this release, it should be reopened or resubmitted with reference to the old defect, depending on the agreement. If the old defect is still open, the tester need not submit a new one.

Tips

- The test manager, defects administrator or intermediary would be well advised to send frequent overviews of found defects to the testers. This keeps the testers abreast of found defects and prompts them to look within their own test for similar defects. Alternatively, the testers could regularly consult the defects administration concerning found defects.

- It may also be agreed that the testers do not look at duplicate defects, to avoid disrupting the progress of the test execution. Checking for duplicate defects is then done by the intermediary, who would be empowered to combine duplicate defects. In cases of doubt, the intermediary should of course consult with the testers involved.

4.7.2.8 Write a defect report

The tester documents the defect in the defects administration by means of a defect report. In this, he describes the defect and completes the necessary fields of the report. The description of the defect should be clear, unambiguous and to the point. The tone should remain neutral, and the tester should come across as impartial, being conscious of the fact that he is delivering bad news. Sarcasm, cynicism and exaggeration are obviously to be avoided.
Ideally, the tester makes clear what the consequences are in the event of the defect not being solved, or what the damage might be in production. This determines the chances of the defect being solved after all. In some cases, the damage is very clear ("Invoices are wrongly calculated") and little explanation is necessary; in other cases, it is less clear ("Wrong use of color in screens") and the tester should clearly indicate what the consequences could be ("Deviation from business standards means that the External Communication department may obstruct release of the application"). Otherwise, it is by no means always possible for the tester to estimate the potential damage, as he lacks the necessary knowledge. The final responsibility for estimating the damage lies with (the representatives of) the users and the client in the defects consultation, which is discussed later.
A difficult question is always how much information the description should contain. The guideline for this is that the defect solver should be reasonably able to solve the defect without further explanation from the tester.

In more detail

'Reasonably' in the above sentence is a difficult concept. The developer would prefer the tester to indicate which statement is wrong in the software. However, this is debugging and comes under the responsibility of the developer. The situation should be avoided in which the tester regularly sits with the programmer to search together for the cause of the defect. This indicates poorly written defects rather than collaborative testing. The tester is at that point no longer involved in testing operations, as the test manager expects of him. If this happens regularly, it will render the plan of the test process unmanageable.

In some cases, the tester finds many small defects in a particular part, e.g. a screen. The inclination is then to keep the administration simple by grouping all these defects into one collective defect report. There is sometimes pressure from the developers to do this, either for the same reason or to make the number of defects appear lower. This is rarely advisable. The chances are that, out of such a collection, a number of defects will be solved in the subsequent release, a number will be solved in the release following, and a number will not be solved at all. Following and monitoring such a collective defect thus becomes an administrative nightmare.

4.7.2.9 Have it reviewed

Before the defect formally enters the defects procedure, the tester has the report reviewed for completeness, accuracy and tone. This may be done by a fellow tester, the test manager, defects administrator or the intermediary. After processing their comments, the defect is formally submitted.

For more information on handling a defect, see [Black, 2004].

4.7.3 Defect report

A defect report is more than just a description of the defect. Other details on the defect need to be established (e.g. version of the test object, name of the tester). In order to do this in a structured manner, a defect report is often divided into several 'fields', in which

the various details can be laid down that are necessary for the management of the defect and for obtaining meaningful information from the administration. The most important reasons for including separate fields, rather than one large free-text field, are:

- The fields compel the defect information to be entered as completely as possible
- It is possible to create reports on selections of defects.

This way it is, for example, easy to select all the outstanding defects, all the defects with the test environment as a cause or all the defects in a particular part of the test object.

Defect reports are almost impossible now without automated support. This may be a simple spreadsheet or database package, but there are also various freeware or commercial tools available. The latter group of tools often has the advantage that the defects administration is integrated with testware management and plan and progress monitoring. Attention should be paid to the matter of authorizations with the tools. It should not be possible for a developer to change or close a tester's defect, but it should be possible for the developer to add a solution to the defect.

Tip

If testers and other parties are geographically far removed from each other, as is often the case with outsourcing or offshoring, it is advisable to purchase a web-enabled defects tool. This allows all the parties to directly view the current status of the defects administration and significantly eases communication on defects.

In more detail

In some organizations, the defects administration is placed within the incidents registration system of the production systems. While this is possible, such a system contains many more information fields than are necessary for a defect. Sometimes this can be adjusted, but sometimes the testers have to learn to deal with the complex system and ignore all the superfluous fields on the screen. This requires decidedly more training time and involves a greater likelihood of incorrect input of defects than with a standard defects administration.

If the defects are stored in an automated administration, a range of reports can be generated. These are very useful for observing certain trends concerning the quality of the test object and the progress of the test process as early as possible. For example, ascertaining that the majority of the defects relates to (a part of) the functional design, or that the defects are concentrated in the screen handling. Such information can be used again for purposes of timely intervention and adopting measures.

The success of the defects administration is determined to a significant degree by the testers' discipline in completing the fields. To this end, the testers should first be sure of the content of each field and how it should be filled in. Particularly in the beginning, there is a need for guidance and monitoring of the completion of defect reports. This is usually a role for the test manager, defects administrator or intermediary, and forms part of the step "Have it reviewed".

The uniformity and consistency of a defect report can be improved by restricting the possible input values for the fields, instead of using free-text boxes. For example, for the cause of a defect, a choice can be made between test basis, test object or test environment. This prevents all kinds of synonyms from being entered ('software', 'code', 'programming', 'program', 'component') that severely obstruct or render impossible any later selection of cause of defect.

First, a description is given below of what a defect report should minimally contain. Subsequently, various recommendations are given as regards expanding on this.

4.7.3.1 Minimum fields in a defect report

A defect report contains the following fields at minimum:

- *Project or system name*
  The name of the (test) project or of the system under test.
- *Unique identification of the defect*
  A unique identity, usually in the form of a (serial) number of the defect report, for purposes of management and tracking progress.
- *Brief characterization*
  A brief characterization of the defect in a limited number of words, maximum one sentence that preferably also clearly indicates the consequence of the defect. This characterization is printed in defects overviews and makes the defect more communicable.
- *Submitter*
  The name of the individual who has submitted the defect.
- *Identification of phase/test level*
  The phase or test level in which the defect was found, e.g. design, development, development test, system test, acceptance test or implementation.
- *Severity*
  The severity category proposed by the tester. This categorization reflects the damage to the business operations. For example:
  - Production-obstructive: involves (high) costs, e.g. because the defect will shut down operations when the system goes into production
  - Severe: (less) costs involved, e.g. because the user has to rework or add items manually
  - Disruptive: little or no costs involved, e.g. chopping of alphanumeric data on the screen or issues relating to user-friendliness
  - Cosmetic: wrong layout (position of fields; colors) which is not a problem for the external client, but can be disturbing to the internal employee.
- *Priority*
  The priority of the solution proposed by the tester. Possible classification:
  - Immediate reworking required, e.g. a patch available within 48 hours that (temporarily) solves the problem. The test process or the current business operations (if it concerns a defect from production) are seriously obstructed
  - Reworking required within the current release. The current process can continue with work-arounds, if necessary, but production should not be saddled with this problem
  - Reworking required eventually, but is only required to be available in a subsequent release. The problem (currently) does not arise in production, or else the damage is slight.

In more detail

At first sight, it does not appear important to make a distinction between severity and priority. These usually run in sync, so that a high level of severity implies a high priority of solving. However, this is not always the case and that is the reason for distinguishing both categories. The following examples illustrate this:

1) With a new release, the internally allocated nomenclature in the software has been amended. The user will not be aware of this, but the automated test suite will suddenly stop working. This is a defect of low severity, but test-obstructive and therefore of very high priority.

2) The user may find a particular defect so disturbing that it may not be allowed to occur in production. This may be, for example, a typo in a letter to a customer. This, too, is a defect of low severity that nevertheless needs to be reworked before going into production.
3) A potentially very serious defect, e.g. the crashing of the application with resulting loss of data, only occurs in very specific circumstances that do not arise often. A work-around is available. The severity level is high, but the priority may be lowered because of the work-around.

- *Cause*
  The tester indicates where he believes the cause to lie, for example:
  TB: test basis (requirements, specifications)
  S: software
  DOC: documentation
  TIS: technical infrastructure.
- *Identification of the test object*
  The (part of the) test object to which the defect relates should be indicated in this column. Parts of the test object may be e.g. object parts, functions or screens. Further detail may be supplied optionally by splitting the field into several fields, so that e.g. subsystem and function can be entered. The version number or version date of the test object is also stated.
- *Test specification*
  A reference to the test case to which the defect relates, with as much relevance to the test basis as possible.
- *Description of the defect*
  The stated defect should be described as far as possible.
- *Appendices*
  In the event that clarification or proof is necessary, appendices are added. An appendix is in paper form, such as a screen printout or an overview, or a (reference to an) electronic file.
- *Defect solver*
  The name of the individual who is solving the defect, has solved it or has rejected it.
- *Notes on the solution*
  The defect solver explains the chosen solution (or reason for rejection) of the defect.
- *Solved in product*
  Identification of the product, including version number, in which the defect should be solved.
- *Status + date*
  The various stages of the defect's life cycle are managed, up to and including retesting. This is necessary in order to monitor the defect. At its simplest, the status levels of "New", "In process", "Postponed", "Rejected", "Solved", "Retesting" and "Done" are used. The status also displays the date.

4.7.3.2 Possible extensions

Besides the above fields, various other fields may be added to the defect report. The advantages of including one or more of the fields below are better management and more insight into the quality and trends. The disadvantages are the extra administration and complexity. Experience shows that the advantages far outweigh the disadvantages in medium-sized and big tests or in cases in which a lot of communication on the defects between various parties is necessary.

- *Identification of the test environment*
  The test environment used, with identification of the starting situation used.

- *Identification of the test basis*
  The test basis used: name of the test basis document, including version number, supplemented if necessary with specific-requirement ¬number.
- *Provisional severity category*
  Provisional: the severity category proposed by the tester.
- *Provisional priority*
  Provisional: the priority of solution proposed by the tester.
- *Provisional cause*
  Provisional: the cause of the defect as estimated by the tester.
- *Quality characteristic*
  The quality characteristic established by the tester, to which the defect relates.

In connection with the solution:
- *Definitive severity*
  The definite category of severity as determined by the defects consultation.
- *Definitive priority*
  The definite priority of solution as determined by the defects consultation.
- *Definitive cause*
  The definite cause of the defect as determined by the defects consultation. Besides the categories mentioned for the minimum defect report, the category of "Testing" is added here.
- *Deadline or release for required solution*
  A date or product release is set, by which the defect should be solved.

In connection with retesting:
- *Retester*
  The name of the tester who carries out the retest.
- *Identification of the test environment*
  The test environment used, with identification of the starting point used.
- *Identification of test basis*
  The test basis used: name of the test basis document, including version ¬number, if necessary supplemented with specific-requirement number.
- *Identification of test object*
  The (part of the) test object that was retested. The version number or version date of the test object is also stated.

In addition, test, defects consultation, retest and comments fields may be added, with which extra information may be optionally supplied, e.g. on corresponding defects or the identification of the change proposal by which the handling of the defect is brought within another procedure.

4.7.4 Procedure

When a defect is taken into the administration, it enters the defects procedure.

Progress of the solving of defects is discussed in a periodic defects consultation. During the preparation and specifying of tests, this consultation is usually held once or twice a week. During test execution, it often increases to once a day. Participants in the consultation are representatives of the parties who submit and/or deal with the defects. From within the testing, this is the test manager, defects administrator or the intermediary. Sometimes a tester is invited to explain a defect. Other parties may be the user organization, functional management, system development and system management. The defects consultation is also sometimes combined with the handling of the change proposals in, for example, a Change Control Board.

Tips

- Conference call
  If the parties are spread over different locations (around the world), this is no reason not to carry out a defects consultation. Conference calls or video conferencing facilitate this.
- Ensure that each participant is well informed of how the defects procedure works and what his or her tasks and responsibilities are. For example, who updates the status of the defects following the defects consultation?

In order of priority, the participants discuss each new defect and decide whether it should be solved, and if so, by whom. In this consultation, the correctness, cause, priority and severity of the defects, as well as the costs of solving them, are discussed. A familiar humorous reaction of developers in this connection is "It's a feature, not a bug". The representative of the testing also has the job of ensuring that the importance of a defect (severity and priority) becomes sufficiently clear to all the parties. The consultation may also request the submitter of the defect to provide additional information or carry out further investigation. The participants in the consultation determine, after carrying out the necessary discussions, the definitive values for cause, priority and severity of a defect.

If the defects consultation agrees that it is a valid defect and the costs of solving it are acceptable, the defect is assigned to a defect solver. If the consultation agrees that it is not a valid defect or that the costs, lead-time or regression risks of solving it are too high, it is rejected. A valid defect that is nevertheless rejected is also known as a 'known error'. In the event of rejection, it may be decided to submit the defect via another channel as a formal change proposal or to devise a procedural solution. Examples of procedural solutions are notes in the help text, instructions to the helpdesk assistants or amendment to the AO procedures. If the consultation does not agree, then the defect is escalated to the decision forum. Representatives of the parties with decision-making powers sit in this forum, such as the client and project manager, who decide on whether or not (and when) the defect is to be solved. The decision forum is not necessarily an independent consultation, but is often the project management meeting or the project board meeting.

The diagram below shows the relationship between the defects consultation and decision forum:

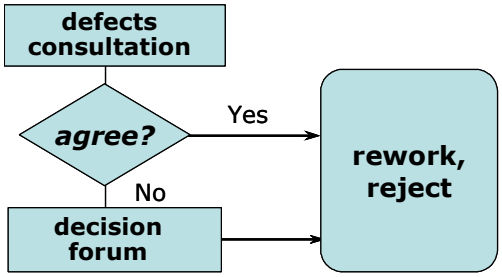


Figure 66. Defects procedure.

The defect solver investigates the defect and solves it. Or it may emerge that the defect has been incorrectly identified as such (a testing mistake) or should be handled by another defect solver. In the latter cases, the defect goes back for discussion. If it is solved, it can be transferred at any time to the test environment to be (re)tested. The tester, preferably the original submitter, carries out the test and checks whether the defect is solved. If so, the defect is closed. If it appears that the defect is not (adequately) solved, then its status is reset and it again undergoes the defects procedure.

The retesting of the defect is an essential step in order to be able to close it. It is unacceptable for the defect solver to solve the defect, test it himself and then close it. Checking whether the defect is solved is the task of the submitter (or his replacement).

In more detail

The time required for researching, submitting, processing, solving and retesting a defect is considerable. Purely administrative and management tasks alone take between one and two hours. This is an important reason to require that the test object be of sufficient quality to enter a test. The pretest is aimed at checking this testability.

Figure 67 shows the life cycle of a defect according to the above procedure, in which the texts in the rectangles show the status of the defect. The diamonds refer to the actors. The dotted line from "Postponed" to "Allocated" means that the defect is postponed in the current release, but should be solved in a future release.

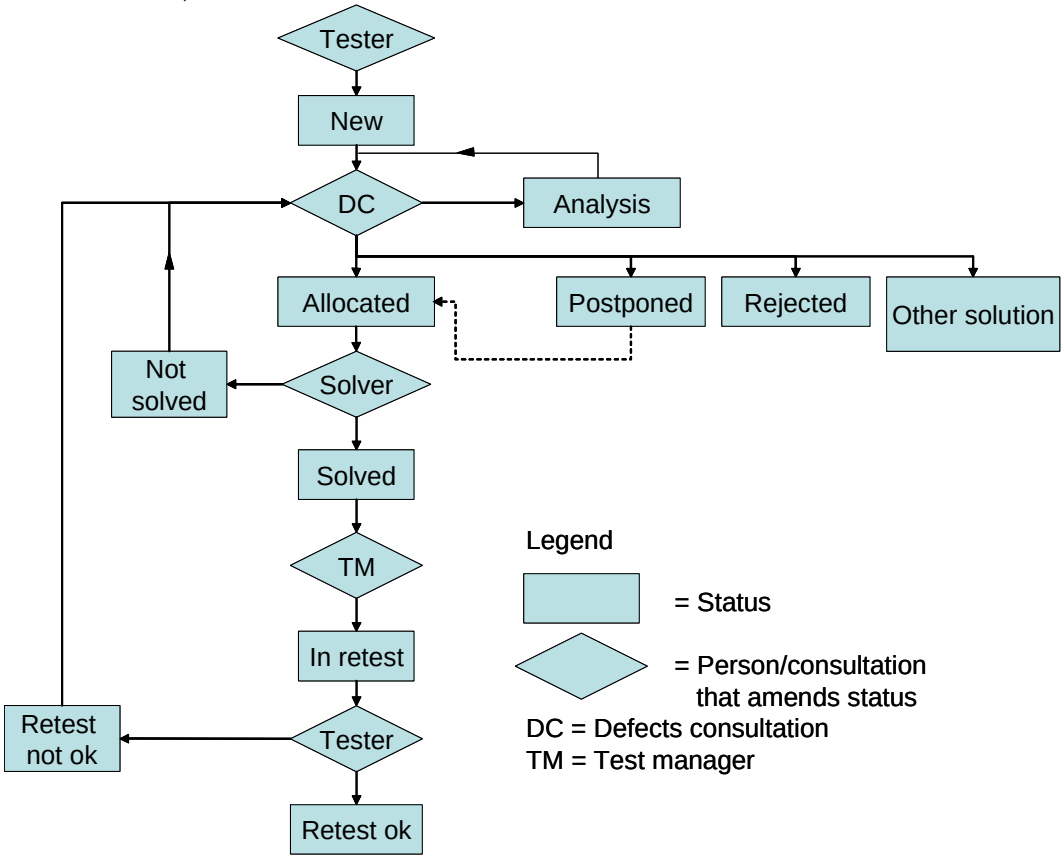


Figure 67. Life cycle of a defect.

4.8 Development tests

4.8.1 Introduction

To allow them to perform at their best in the market, users of information systems demand ever-faster delivery of the systems as well as more flexibility. Development methods are increasingly geared to follow changing requirements closely, even in the midst of a project. The architectures and development environments that make all this possible are

becoming more complex and bigger in scale. This changes the development requirements.

The growing demand on the developer: to deliver the right quality, on time and right first time!

In addition to increased knowledge of development languages, methods, environments and architectures, this also calls for deeper knowledge of quality delivery. Difficult questions in this connection are: what is the quality level required for the client, and how can this be realized and demonstrated through testing? Individual interpretation of the required quality and random testing provide no guarantee of eventual success. Predictable and proven quality of the delivered software gives the project or the department the opportunity to organize the subsequent test levels, such as the system test and the acceptance test, more efficiently. A reduction in the number of redeliveries and retests in those test levels, in particular, delivers significant time savings. In order to realize the higher quality of software, increasingly high demands are placed on the development tests, and development testing is fast becoming a mature part of the entire testing process.

Soon the time will be over that barely any requirements are being set as regards the development tests, and the (increasingly mature) system and acceptance tests are being relied upon to rectify the lack of quality for going into production. The resulting lengthy and costly reworking and retesting cycles have become unacceptable to most organizations.

This section discusses the development tests, makes a comparison with the other test levels and describes specific test tools for development testing. Various quality measures are also discussed that can be used in, or can influence, the development tests. An important measure here is the concept of selected quality. After that, this section describes the activities of development testing according to the TMap life cycle model.

While developers and development testers are a logical target group for this chapter, they should not expect, after reading this one chapter, to know all about how to organize and execute development tests. The target groups of this chapter are:

- The developer/development tester, for ideas on a better development test
- The test consultant who is asked to support (the organization of) the development testing
- The test manager for the overall test process who has to coordinate the development tests with the other test levels
- The line manager or project manager of the developers who is interested in improved control over the quality of the software produced, and who wants to know how this can be achieved.

4.8.2 Development testing explained

This section consists of a number of subsections. These are, in sequence:

- What is development testing?
- Characteristic
  With a focus on how they differ from system tests and acceptance tests
- Advantages and disadvantages of improved development tests
- Context of development testing
  The influence of the development method and technical implementation
- Unit test
- Unit integration test

4.8.2.1 What is development testing?

Development testing is understood to mean testing using knowledge of the technical implementation of the system. This starts with the testing of the first/smallest parts of the system: *routines, units, programs, modules, components, objects, etc*. Within TMap, the term 'unit' and therefore unit test is used exclusively in this context.

When it has been established that the most elementary parts of the system are of sufficient quality, larger parts of the system are tested integrally during the unit integration tests. The emphasis here lies on the data throughput and the interfacing between the units up to subsystem level.

Definition

Unit Test (UT)
The *unit test* is a test carried out in the development environment by the developer, with the aim of demonstrating that a unit meets the requirements defined in the technical specifications.

Unit Integration Test (UIT)
The *unit integration test* is a test carried out by the developer in the development environment, with the aim of demonstrating that a logical group of units meets the requirements defined in the technical specifications.

4.8.2.2 Characteristics

A pitfall in organizing development tests is the temptation to set up the test process from the viewpoint of a system test or acceptance test. For when development tests are compared with the system test and the acceptance test, a number of significant differences come to the fore:

- In contrast to the system test and acceptance test, the development tests cannot be organized as an independent process with a more or less independent team. The development tests form an integral part of software development, and the phasing of the test activities is integrated with the activities of the developers.
- Because development testing uses knowledge of the technical implementation of the system, other types of defects are found than those found by system and acceptance tests. It may be expected of development tests, for example, that each statement in the code has been touched on. A similar degree of coverage is, in practice, very difficult for system and acceptance tests to achieve, since these test levels focus on different aspects. It is therefore difficult to replace development tests with system and acceptance tests.
- With the unit tests, in particular, the discoverer of the defects (i.e. the tester) is often the same individual who solves them (i.e. the developer). This means that communication on the defects may be minimal.
- The approach of development testing is that all the found defects are solved before the software is transferred. The reporting of development testing may therefore be more restricted than that of system and acceptance testing.
- It is the first test process, which means that all the defects are still in the product, requiring cheap and fast defect adjustment. In order to realize this, a flexible test environment with few procedural barriers is of great importance.
- Development tests are often carried out by developers themselves. The developer's basic intention is to demonstrate that the product works, while a tester is looking to demonstrate the difference between the required quality and the actual quality of the product (and actively goes in search of defects). This difference in mindset means that

sizeable and/or in-depth development tests run counter to the developer's intention and, with that, meet with resistance and/or result in carelessly executed tests.

In more detail

Figure 68 [Pettichord, 2000] sums up a number of salient characteristics of testers and developers:

Testers	Developers
Get up to speed quickly	Thorough understanding
Domain knowledge	Knowledge of product internals
Ignorance is important	Expertise is important
Model user behaviour	Model system design
Focus on what can go wrong	Focus on how it can work
Focus on severity of problem	Focus on interest in problem
Empirical	Theoretical
What's observed	How it's designed
Skeptics	Believers
Tolerate tedium	Automate tedium
Comfortable with conflict	Avoid conflict
Report problems	Understand problems

Figure 68. Characteristics of testers and developers.

4.8.2.3 Advantages and disadvantages of improved development tests

In practice, development testing is often unstructured: tests are not planned or prepared; no test design techniques are used and there is no insight into what has or has not been tested or with what test intensity. With that, insight is also lacking into the quality of the (tested) product. Often during the system and acceptance tests, there are lengthy and inefficient cycles of test/repair/retest in order to get the quality up to an acceptable level. It therefore stands to reason that development testing should be better organized. A number of arguments are presented below as to why this does not take place in practice (arguments against) and why it is important that it should take place (arguments for).

Arguments against

The most important arguments as to why the need for more structure and thoroughness in development testing is *not* self-evident are:

- *Pressure of time / not cost-effective*
  Developers are often under severe pressure of time. The priorities of the development team are defined by the criteria by which it is judged. Assessment is usually made based on hard criteria, such as lead-time and delivered functionality. Assessment by a much softer criterion, such as quality, is more difficult and is therefore rare in practice. A developer who is committed to a completion date will either communicate openly and honestly when things are not going smoothly, or give less time to his own testing if the coding is in trouble. From the point of view of personal performance (and assessment), the latter is not unthinkable. After all, benefits to a development team of thorough testing are relatively small, even though they are many times greater for the project as a whole.
- *Sufficient faith in the quality*
  A developer is usually proud of his product and considers it to be of good quality. It is therefore not logical as a developer to expend a lot of effort in finding fault with his own product.

- *There will be another thorough test to follow*
  In the subsequent phase, e.g. the system test, a much more intensive test will be carried out than development testing can ever do. Why should the development tester then pay much attention to more and better testing, when it is to take place later more extensively?

Arguments in favor

The most important argument for more structure and thoroughness in development testing is that it enables the developer to establish for himself that the software is of sufficient quality to be delivered to the next phase, probably the system test. The meaning of "sufficient quality" is of course open to discussion. Below is indicated that "sufficient quality" has many advantages for *the development team*:

- Less reworking will be necessary after delivery, since the products that are delivered to the subsequent phase are of higher quality.
- The planning is better, since the often uncertain volume of rework declines.
- The lead-time of the total development phase is, for the same reason, shorter.
- Reworking as early as possible is much cheaper than at a later stage, since all the knowledge of the developed products is still fresh in the memory, whereas by the later stage people have often already left the development team.
- Analyzing defects you find yourself is much faster and easier than analyzing defects found by others. The more distance (both organizational and physical) the finder has, the more difficult and time-consuming the analyzing often is. Even more so, since in later phases the system is tested as a whole and the found defect may be located in many separate components.
- The developers get faster feedback on the mistakes they make, so that they are better able to prevent similar mistakes in other units.
- Certain defects, particularly on the boundaries of system functionality and underlying operating system, database and network, can best be detected with development tests. If the development testing finds too small a proportion of these defects, this will have consequences for the system and acceptance tests, which then have to produce a disproportionate effort (in the detection of such defects), using inefficient techniques, in order to achieve the same quality of the test object had the development tests been adequately executed.

These advantages apply for the total project, and even for the total life cycle of the system *to a greater degree*, because the later test levels also benefit from these advantages (often even more so!), for example because much fewer retests are necessary. Accordingly, the advantages of a more structured development test¬ approach far outweigh the disadvantages. However, a necessary condition for successful structuring of the development testing is that the various parties involved, such as the client, the line and project manager and the developers, are aware of the importance of a better test process. For example, the project manager should assess the development team much more on delivered quality than simply on time and money. The development department may also set requirements on all the executed development tests. Each development test in an individual project should at least meet these requirements.

4.8.3 Context of development testing

Development testing bears a very close relationship with the development process and cannot really be considered separately from it. Much more knowledge of the technical implementation of the system or package is required as far as development testing is concerned than for a system or acceptance test. In order to organize the development test well, allowance must certainly be made for the development process used and the technical implementation.

Tip

Ensure that, as adviser or test manager in the organization of the development testing, you have sufficient knowledge of the development process used and the technical implementation. This will also make you a useful partner in the dialogue with the developers, without having to be an expert.

4.8.3.1 Influence of the development method

Roughly three streams of development methods can be distinguished: waterfall, iterative and agile.

- Waterfall, which includes the following characteristics: the development of a system in one go, phased with clear transfer points, often a long cyclical process (including SDM)
- Iterative, characterized by: incremental development of the system, phased with clear transfer points; short cyclical process (iterations) (including DSDM and RUP). Iterative methods take up an intermediate position between waterfall and agile.
- Agile, characterized by four principles: individuals and interaction above processes and tools, working software above extensive system documentation, user input above contract negotiation, responding to changes above following a plan (including eXtreme Programming and SCRUM).

To discover what influence the development method has on (the organization of) the development testing, it should be considered to what degree the following aspects play a role:

- Instructions for development test activities
  Many methods go no further than indicating that development tests need to be carried out. Structured guidelines are seldom supplied. Extreme Programming (XP), as one of the agile methods, is a positive exception in this area. Three of the most important practices in development testing are Pair Programming, Test-Driven Development and Continuous Integration.
- Quality of the test basis
  The waterfall method is usually established in a formally described form. With iterative and agile development methods, the form of the test basis is much less formal and often agreed verbally (through consultation with users). This means that it is more difficult with iterative and agile methods to discover all that requires to be tested. For example, the fault handling and exceptional situations (together estimated to be as much as 80% of the code) are often under-exposed in such forms of test basis. Greater reliance is placed on the expertise and creativity of the development testers as regards devising and executing tests for these
- Long- or short-cyclical development
  With short-cyclical development, proportionately more time is spent on testing, particularly due to the need to execute a much more frequent regression test (every cycle at minimum) on the system so far developed.

4.8.3.2 Influence of the technical implementation

Over the years, the IT world has grown into a patchwork quilt of technological solutions. To represent this simply, you could say that the first systems were set up as monoliths, meaning that the presentation, application logic and information storage were one giant whole. Some of these systems have been in operation for more than 30 years now. The monolithic systems were followed by systems based on client/server architectures. Then came the 3-

layer systems with separate presentation, application logic and database layers. In parallel with this, obviously, there was the rise of the big software packages, such as SAP, and of Internet and browser-based applications. These days, many systems are set up in distributed fashion, which means that they consist of different, often physically dispersed, components or services, while the system is still seen by the outside world as a cohesive whole, owing to close collaboration.

The systems were developed with the aid of a large arsenal of programming languages, whether or not object-oriented, in development environments that support (automated) testing to a greater or lesser degree.

As indicated in section 4.1.3, "The essentials of TMap NEXT®", testing is a risk-based activity, in which risk = chance of failure x damage, with chance of failure = frequency of use x chance of a fault. The relevance of the above summary of 50 years of system development in one paragraph is that the technical implementation determines to a great degree the type of faults that can be made and in which parts the chances of faults are the greatest. The test strategy of development testing is thus strongly dependent on the technical implementation, more so than the system and acceptance tests, where more attention is paid to the specifications of the system and the potential damage.

In more detail

The increasing use of distributed systems with large numbers of components and services places high demands on the quality of the individual components or services. The complex collaboration between all these components and services makes the finding of the source of a defect very difficult and time-consuming. The result of integrating qualitatively inadequate components or services into the system and hoping that the defects will be found by the system or acceptance tests will be that the required system quality (on time and within budget) cannot be delivered. The technical nature of many components and services means that the development tests bear a heavier responsibility for establishing that the separate components or services are of sufficient quality *before* they are integrated.

4.8.4 Unit test

In unit testing, it is important to realize what the place of testing is within development. The unit testers are usually the developers, who test their own unit. The development project leader, a separate test coordinator or the application integrator coordinate the tests.

A point to note is the specifying of test cases. Developers do not always see the usefulness of this. By opting where possible for 'light' test design techniques and elementary forms of test documentation in particular, the degree of acceptance is considerably increased. Particularly with manual unit tests, considerable powers of persuasion are necessary to convince the developers that the writing out of test cases in those specific instances offers advantages over the unprepared execution of the tests.

In more detail

A good example that shows the advantage of test design techniques and the specifying of test cases is the testing of a multiple condition (IF A=1 and B=2 and C=3 THEN …). With the aid of the test design technique Elementary Comparison Test (ECT), it is relatively simple to derive a limited set of test cases (4 in this example) that provide a high degree of test coverage. Deriving test cases without a technique here quickly leads to either too meager test coverage or a multiple of test cases (8 in this example).

More and more development environments are now making it possible to include the (automated) test code in the (source) code. The unit test then consists of starting the test code, which subsequently executes (a part of) the source code. Such unit tests are grouped into a 'test harness'.

Definition
A *test harness* is a collection of software and test data configured for a development environment with the purpose of testing one unit or a series of units, whereby the behaviour and output are checked.

The writing of unit tests in a test harness is an extra effort that should not be ignored. Experience teaches us that the developing of test code costs 10%-30% extra effort [Vaaraniemi, 2003].

Development methods have firmly embraced the possibility of including test code directly with the (source) code. Initiatives like Test-Driven Development (see section 7.2.7) make testing an increasingly important part of system development.

4.8.5 Unit integration test

When a unit has been tested and approved, it can be integrated with other units into a working (part of the) system. Rarely are all the units combined and tested at one time – the so-called "big bang" scenario. The disadvantage of this late integration is that, in general, many defects are found, and tracing the causes takes up a lot of time. A more effective method is integrating numbers of units together in steps and testing after each integration step. In this way, defects are found at an early stage, when the cause is still relatively easy to detect. Unit integration testing thus plays a role particularly in repeatedly demonstrating that the new or amended unit(s) continue to work well in conjunction with earlier integrated units.

The best sequence of integration and the number of integration steps required depends on the location of the most risk-related parts of the system. Ideally, the integration starts with those units in which the most problems are expected. This prevents serious problems arising at the end of the unit integration test.

Executing unit integration tests requires extensive knowledge of the content, structure and especially the information exchange of underlying units. This in-depth knowledge means that often a separate role is allocated to the integration of units: the application integrator.

The developments in the area of development environments also facilitate automated compilation, integration and testing. This takes place with the aid of 'build & deploy' scripts. 'Build' in this context is the combining of the various software components into a software package that can be exported to a particular environment. 'Deploy' is the rolling out of the software in the target environment, in other words the conversion of the software package into the operational (installed) form. Scripts make it possible to execute build & deploy by automation. Within the build & deploy scripts, the test harness is called up. In this, besides the automated unit tests, tests are also built and executed that exceed the boundaries of the units and the integration tests. Integration test cases often form a functional path from beginning to end through the application. By making use of stubs and drivers tests can be included at an early stage that run through the application from beginning to end. As with automated execution of unit tests, this possibility of automatic integrating and testing has found its way into the development methods. Rather than seeing the integration (test) as mainly a concluding activity, the Continuous Integration

method has been introduced, which brings to the fore as much as possible any problems in connection with the combining of units.

4.9 Estimating the test effort

Estimation techniques

There are various techniques to create an estimate. This chapter begins with an explanation of the different levels at which estimates can be done and an overview of the suitable techniques for estimating specific quality characteristics. The following estimating techniques are then discussed:

- Estimation based on ratios
- Estimation based on test object size
- Work Breakdown Structure
- Evaluation estimation approach
- Proportionate estimation
- Extrapolation
- Test point analysis

Estimating

Estimates can be made at various levels, as shown in figure 69.

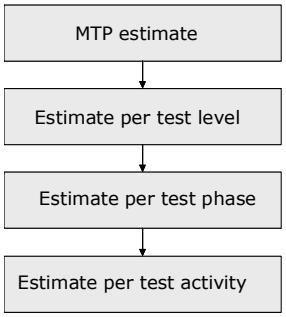


Figure 69. Estimation levels.

Estimates for a MTP are created early on in the project. Often, not all knowledge of the test object is available at this point. As a consequence, the accuracy of the estimate is limited. The size and complexity of the test object may change during the project. It is important for the test manager to make it clear to the stakeholders that the estimate is based on a number of assumptions and therefore details will have to be added later. A possible solution is to use margins to represent the initial estimate for an MTP.

The estimate in the MTP constitutes the framework for the estimates per test level (e.g. system test, user acceptance test, and production acceptance test). The required time for the various phases – Control, Setting up and maintaining infrastructure, Preparation, Specification, Execution and Completion – is then established for the test level. Separate test activities are estimated within the test phases. The time necessary to create the MTP (Planning) is not included in the estimates. A fixed number of hours is usually estimated for this. After all, establishing the plan consists of executing clearly defined activities. The impact of e.g. the test object size on the time required to create the MTP is limited in this context. If there is an impact, it will be noticeable in particular during the activities "Analyzing product risks" and "Determining test strategy". In practice, some 60 to 160 hours are usually invested in creating the MTP.

As the estimate is made later in the test process and therefore at a lower level, more knowledge of the test object is available. Moreover, experiences from earlier on in the process can be used, making the estimate more accurate.

Independent of the level, creating the plan consists of the following generic steps:
Inventory the available material that can serve as a basis for the estimate:
1. Inventorize the available material that can serve as a basis for the estimate.
2. Select (a number of) estimating techniques.
   It is recommended to use multiple techniques in parallel. This makes it possible to compare the outcome of the various techniques. In addition to estimating techniques, it is worthwhile asking an experienced employee to make an estimate of the required time (expert estimate).
3. Determine the definitive estimate.
   The aim of this step is to combine the outcomes of the previous step into one single estimate. If the outcomes vary little, taking an average will work. In other cases the differences have to be analyzed. If an adequate estimate cannot be made after analyzing the differences, the client must be consulted. The test manager explains the problems and makes proposals to achieve a correct estimate.
4. Present the outcome.
   The aim of presenting the outcome is to provide insight to the business into the consequences of the selected test strategy and approach. It is important to show clearly which assumptions were made. Especially with an estimate created very early on in the process, assumptions will be involved that will become more concrete later on in the process.

As discussed earlier, there are various estimating techniques to create an estimate. Choosing the right ones in particular is a step requiring experience. The sections below describe the estimating techniques, based on the following principles:
- Estimating the test activities in the development phase (unit test and unit integration test) is an integrated component in estimating the realization project and is not taken into consideration unless explicitly specified.
- Where possible, experience figures are mentioned for the specified techniques. We explain the background of these figures. The figures shown must always be considered within the described context. They do not necessarily apply in a different situation.
- One retest is included in all of the experience figures mentioned in subsequent sections.

Please refer to section 4.11, "Metrics", for the structured collection and analysis of test estimating figures.

An adequate choice from the various techniques can be made with the use of two tables. These tables answer the following two questions:
- Which technique is suitable for which level of estimating?
- Which techniques are suitable for estimating which quality characteristics?

The answers to these questions are shown in the tables below.

	Master test plan	Detailed test plan	Test phase	Test activity
Estimating based on ratios	X	X	X	(X)
Estimating based on size	X			
Work Breakdown Structure (WBS)	X	X	X	X
Evaluation estimating techniques	X			
Proportionate estimation	X	X	X	(X)

	Master test plan	Detailed test plan	Test phase	Test activity
Extrapolation			X	X
Test point analysis (estimating based on size and strategy)	X	X		

The possible estimating techniques are shown per quality characteristic in the table below. The table distinguishes between three different levels of test intensity for explicit tests, i.e. •, •• and ••• (low, medium and high).

	Evaluation	UT	UIT	Implicit test	Explicit test	Explicit test	Explicit test
Test intensity ⇒	•	•	•		•	••	•••
Quality characteristic ⇓	No. of pages [1]	[2]	[2]	[3]			
Connectivity	Tpa-s				-	-	-
Continuity	Tpa-s				Timebox[7] week	Timebox[7] month	Timebox[7] quarter
Data controllability	Tpa-s				-	-	-
Effectivity	Tpa-s				Tpa[6]	Tpa[6]	Wbs
Efficiency	Tpa-s				-	Wbs	-
Flexibility	Tpa-s				-	-	-
Functionality	Tpa-s	Hour box[7]	Hour box[7]		Tpa	Tpa	Tpa
Infrastructure	Tpa-s				-	-	-
Manageability	Tpa-s[4]				Wbs[5]	-	-
Maintainability	Tpa-s				-	-	-
Performance Batch Online	Tpa-s				Tpa Wbs	Tpa Wbs	Tpa Wbs
Portability	Tpa-s				Wbs	Tpa	Tpa
Reusability	Tpa-s				-	-	-
Security	Tpa-s				Tpa	Tpa	Wbs
Suitability	Tpa-s				Tpa[6]	Tpa[6]	Tpa
Testability	Tpa-s				-	-	-
User-friendliness	Tpa-s				Wbs	Wbs	Wbs

Notes on the table:
- It is not possible to indicate a specific estimating technique for this level of test intensity.

1) Several pages must be read when evaluating intermediate products on quality characteristics. Quality characteristics that have to do with functionality require a study of the pages on which the functionality is described. Other quality characteristics are generally described on other pages. This results in a varying number of pages per quality characteristic for verification.
2) It is assumed that the estimate of the standard test activities in the UT and UIT is part of the estimate of the realization. If desirable, extra attention to testing during the UT and

UIT can be specified. The estimating technique for this is an hour box, in which e.g. a supplement rate is added to the build effort (e.g. 10%) or part of the effort for the ST.
3) TPA-i is the component for implicit testing of a quality characteristic during the testing of another quality characteristic. In TPA, this results in an additional supplement of 0.02 when determining the Q_d.
4) TPA-s is the evaluation component of TPA.
5) WBS = Work Breakdown Structure.
6) If effectivity and suitability are tested with the same test type/test technique, the effort is included once.
7) The time box and hour box are determined by factors outside the test process. Time box week in the table above means that testing takes a period of one week.

4.9.1 Estimation based on ratios

To use ratios as a basis to create an estimate, it is important to collect the greatest possible amount of experience figures. This makes it possible to derive 'standard' ratios for similar projects. Similar projects are projects that are the same in terms of certain key properties. For instance the same development method, the same development platform, the same software environment, the same experience level of the developers, etc.

Naturally, the own ratios of an organization generally are the best ones to use within that organization. Ratios can be used at all estimation levels. At the level of test activities in test phases however, the ratios are so specific that they can only be used within one organization and often even within the area of application (project or system).

Below please find a number of ratios between tests and other development activities from actual practice. An organization can use these observations as a starting point. By then keeping track of its own experience figures, the organization can match the ratios more and more adequately to its own practice.

The various observations are based on the following standardization of terms:
- Functional design (FD) = functional detailed design.
- Realization, consisting of the technical design (TD), programming (P), unit and unit integration test (UT and UIT).
- Functional test. This concerns the testing of the functionality quality characteristic, with the FD as the test basis. The ST and AT test levels are used for this purpose.

Observed ratios in an average risk profile are as follows:
- FD : Realization : Functional test = 2 : 5 : 3
  In an environment with a formally complete FD, waterfall development method, 3GL programming language, and a structured test method of operation. These figures were found to apply for the activities in the maintenance phase as well, with testing only involving a test of the change.
- (FD+TD) : (P + UT + UIT) : Functional test = 1 : 3 : 3
  In an environment with an incompletely detailed FD, experienced builders who fill the FDs themselves, and a starting test method of operation.
- FD : Realization : Functional test = 1 : 2 : 1.2
  In a test environment with a formally complete FD, waterfall development method, experienced builders, and a functional test that does not have maximum test coverage but is driven by risk, and a maximized budget. The method of operation is structured.

Within a test level, ratios can be used to estimate the various phases. Here, too, observations from actual practice are available:

- For a system test with good but complex specifications, the observed ratio is as follows: Preparation 6%, Specification 54%, Execution 21%, Completion 2%, and 17% for Control and Setting up and maintaining infrastructure taken together.
- The following ratio was observed for a system with an inadequate test basis: Preparation 21%, Specification 33%, Execution 24%, Completion 5%, and 17% for Control and Setting up and maintaining infrastructure taken together.

Note: in both cases, 160 hours were spent on creating the MTP.

4.9.2 Estimation based on test object size

The size of the test object can be established in different ways. The term Test Object Size Meter (TOSM) is used to indicate the size of a test object in a uniform manner. Based on a test object size determined this way, the following number can be used to estimate the functional test even without the strategy being known (yet).

1.5 to 4 hours per size unit (TOSM)

The actual number for a specific area of application depends on:
- type of environment (web, database)
- support provided
- quality of the test basis
- size of the project, towards factor 2 for very small and very big projects
- required reporting
- experience of testers.

Organizations can maintain experience figures to make ever more reliable estimates.

The size of a test object (and therefore the number of TOSMs) can be established in the following ways:
- Detailed functional description
  A function point analysis can be performed on a detailed functional description (e.g. a functional design). The result of the function point analysis is a number of function points (FP). One function point is then equaled to one TOSM, making the size of the test object (= number of TOSMs) the same as the number of function points.
- Data model
  If a data model is available, the following approach can be used to establish the size of the test object: determine the number of logical data collections (LDCs) and estimate the complexity. The size of the test object is found by multiplying the number of data collections by the value in the table below.

No. of LDCs	Complexity		
	Low	Medium	High
< 10	25	28	35
10 - 25	28	35	42
> 25	35	42	47

- Requirement pages
  The literature contains experience figures to relate the size of the test object to the number of requirement pages. Generally speaking, this means that not all information concerning the conditions under which the data were measured is available.
  - 1 A4-sized page of requirements without diagrams = 15 TOSMs [Collard, 1999].

- In a large classical project in which a highly detailed functional design without illustrations was available, the following experience figure was measured:
- 1 A4-sized page of requirements = 2.5 to 3 TOSMs.

- Number of screens
  If the number of screens is a determinant for the size of the application, the following derivation can be used [Collard, 1999]:
  1 screen (window/webpage) = 8 TOSMs.
- Program source code
  For a new development project, clearly the program source code is not available until after the realization process. For a migration or maintenance project, for instance, the derivations below may be applicable:
  - 1 kilo lines of code (3 GL) = 17 TOSMs [Collard, 1999].
  - [Capers Jones, 1996]

1 KLOC (kilo lines of code)	Number of TOSMs
C	6,6
Algol, Cobol, Fortran	10
PL/1	12
Lisp, basic	16
4GL database	25
Objective C	39
Smalltalk	49
Query languages	60

4.9.3 Work Breakdown Structure

The Work Breakdown Structure (WBS) is an estimating approach based on splitting up the activities into partial activities up to a level of detail at which the required time per activity can be estimated. By adding the time required for the partial activities, the total required time is calculated.

The table below shows the number of hours per quality characteristic. For quality characteristics where the strategy matters, this is shown. The hours are derived from actual practice. Please note that the experience base and therefore the how hard the figures are differs. Levels of hardness are:

- Hard: experience from multiple projects, confirmed on the basis of multiple sources
- Experience: based on a few sources
- Soft: an estimate by experienced test consultants.

Practice demonstrates which factors have the greatest impact on the definitive number of hours. These factors are shown.

Quality characteristic	Strategy	Hardness estimation	Hours	Important factors for variation in size
Manageability Installability		soft	24	
Security	•••	experience	80	Minimal, hour box
Effectivity	•••	soft	350	Incl. hours of users
Continuity	•••	N/A		Depends on the duration of shadow production

Quality characteristic	Strategy	Hardness estimation	Hours	Important factors for variation in size
User-friendliness	•	hard	70	Size of application (limit 15/100 screens) Scope of research question (limit: several subjects)
User-friendliness	••	hard	80	
User-friendliness	•••	hard	130	
Performance, online	••	hard	192 tot 224	Low: 15 user tasks High: 40 user tasks Complex database
Portability	•	soft	28	
Efficiency	••	soft	28	

Please note: The table above does not include hours for e.g. setting up a usability lab or selecting test-support packages. The starting point is that the required facilities must be available.

4.9.4 Evaluation estimation approach

One of the size bases for evaluations often mentioned in the literature is the number of pages of the document that is being evaluated.

Figures from the literature: 1-4 pages per hour per evaluator per size unit, depending on:
- the number of quality characteristics looked at
- the evaluator's experience
- the required depth
- the formality of the evaluation type – the more formal the evaluation, the more time it takes.

4.9.5 Proportionate estimation

This estimating technique is based on a total quantity of budget to test the entire test object. The total amount is divided over the distinguished components. When dividing the total budget over the various components, the allocated risk class (for a test strategy) and the size of the components are taken into account. A factor is chosen for each risk class (in the test strategy) that enables a weighted distribution. For example:
- Risk class A is allocated a factor 1.5
- Risk class B is allocated a factor 1
- Risk class C is allocated a factor 0.6.

The steps to derive an estimate are as follows:
1. Calculate the product of the size of the object part to be tested with the factor associated with the risk class of that object part. Do this for all object parts.
2. Add the outcomes from step 1.
3. Determine the scaling factor by dividing 100 by the result of step 2.
4. Calculate the number of hours per object part by multiplying the results of step 1 by the scaling factor.

An example to clarify this

100 hours are to divided over 5 object parts. The size and a risk class have been determined for each object part. The number of hours per object part is then established following the steps above.

Object part	Size	Risk class	Factor	Size x Factor	Scaling factor	Number of hours
1	10	C	0.6	6		7.86
2	15	A	1.5	22.5		29.48
3	7	B	1	7		9.17
4	25	A	1.5	37.5		49.12
5	5	C	0.6	3		3.93
Total				76	100/76=1.31	100 (100.56)

4.9.6 Extrapolation

In this estimating method, measurements are made as early on in the project as possible to build experience figures. Once it is known what percentage of the work was done in how much time, it can be established (on approximation) how much time is required for the remainder of the work.
This method is used a lot in practice to estimate test activities within a test level. It is also very suitable to estimate test activities in incremental development methods.

4.9.7 Test point analysis

This section describes the test estimating technique test point analysis (TPA). Test point analysis makes it possible to estimate a system test or acceptance test in an objective manner. Development testing is an implicit part of the development estimate and is therefore outside the scope of TPA. To apply TPA, the scope of the information system must be known. To this end, the results of a function point analysis (FPA) are used. FPA is a method that makes it possible to make a technology-independent measurement of the scope of the functionality provided by an automated system, and using the measurement as a basis for productivity measurement, estimating the required resources, and project control. The productivity factor in function point analysis does include the development tests, but not the acceptance and system tests.

Test point analysis can also be used if the number of test hours to be invested is determined in advance. By executing a test point analysis, any possible risks incurred can be demonstrated clearly by comparing the objective test point analysis estimate with the number of test hours determined in advance. A test point analysis can also be used to calculate the relative importance of the various functions, based on which the available test time can be used as optimally as possible. Test point analysis can also be used to create a global test estimate at an early stage.

4.9.7.1 Philosophy

When establishing a test estimate in the framework of an acceptance or system test, three elements play a role (see figure 70 "Estimating basic elements"):

- The size of the information system that is to be tested.
- The test strategy (which object parts and quality characteristics must be tested and with what thoroughness, what level of test intensity?).
- The productivity.

The first two elements together determine the size of the test to be executed (expressed as test points). A test estimate in hours results if the number of test points is multiplied by the productivity (the time required to execute a specific test intensity level). The three elements are elaborated in detail below.

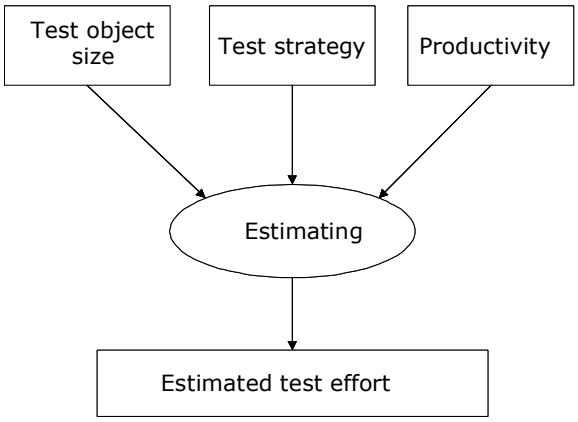


Figure 70. Estimating basic elements.

Size

Size in this context means the size of the information system. In test point analyses the figure for this is based primarily on the number of function points. A number of additions and/or adjustments must be made in order to arrive at the figure for the test point analysis. This is because a number of factors can be distinguished during testing that do not or barely play a part when determining the number of function points, but are vital to testing. These factors are:

- Complexity
  How many conditions are present in a function? More conditions almost automatically means more test cases and therefore greater test effort.
- System impact
  How many data collections are maintained by the function and how many other functions use them? These other functions must also be tested if this maintenance function is modified.
- Uniformity
  Is the structure of a function of such a nature that existing test specifications can be reused with no more than small adjustments. In other words, are there multiple functions with the same structure in the information system?

Test strategy

During system development and maintenance, quality requirements are specified for the information system. During testing, the extent to which the specified quality requirements are complied with must be established. However, there is never an unlimited quantity of test resources and test time. This is why it is important to relate the test effort to the expected product risks. We use a product risk analysis (section 2.6) to establish, among other things, test goals, relevant characteristics per test goal, object parts to be distinguished per characteristic, and the risk class per characteristic/object part. The result of the product risk analysis is then used to establish the test strategy. A combination of a characteristic/object part from a high risk class will often require heavy-duty, far-reaching tests and therefore a relatively great test effort when translated to the test strategy. The test strategy represents input for the test point analysis. In test point analysis, the test strategy is translated to the required test time.

In addition to the general quality requirements of the information system, there are differences in relation to the quality requirements between the various functions. The reliable operation of some functions is vital to the business process. The information system was developed for these functions. From a user's perspective, the function that is used intensively all day may be much more important than the processing function that runs at night. There are therefore two (subjective) factors per function that determine the test intensity: the user importance of the function and the intensity of use. The test intensity, as it were, indicates the level of certainty or insight into the quality that is required by the client. Obviously the factors user importance and intensity of use are based on the test strategy.

The test strategy tells us which combinations of characteristic/object part must be tested with what thoroughness. Often, a quality characteristic is selected as characteristic. The test point analysis also uses quality characteristics, which means that it is closely related to the test strategy and generally is performed simultaneously in actual practice.

Tip

Linking TPA parameters to test strategy risk classes
TPA has many parameters that determine the required number of hours. The risk classes from the test strategy can be translated readily to these parameters. Generally, the TPA parameters have three values, which can then be linked to the three risk classes from the test strategy (risk classes A, B and C).
If no detailed information is available to divide the test object into the various risk classes, the following division can be used:

- 25% risk class A
- 50% risk class B
- 25% risk class C.

This division is then used as the starting point for a TPA.

Productivity

Using this concept is not new to people who have already made estimates based on function points. Productivity establishes the relation between effort hours and the measured number of function points in function point analysis. For test point analysis, productivity means the time required to realize one test point, determined by the size of the information system and the test strategy. Productivity consists of two components: the skill factor and the environment factor. The skill factor is based primarily on the knowledge and skills of the test team. As such, the figure is organization and even person-specific. The environment factor shows the extent to which the environment has an impact on the test activities to which the productivity relates. This involves aspects such as the availability of test tools, experience with the test environment in question, the quality of the test basis, and the availability of testware, if any.

4.9.7.2 Global method of operation

Schematically, this is how test point analysis works:

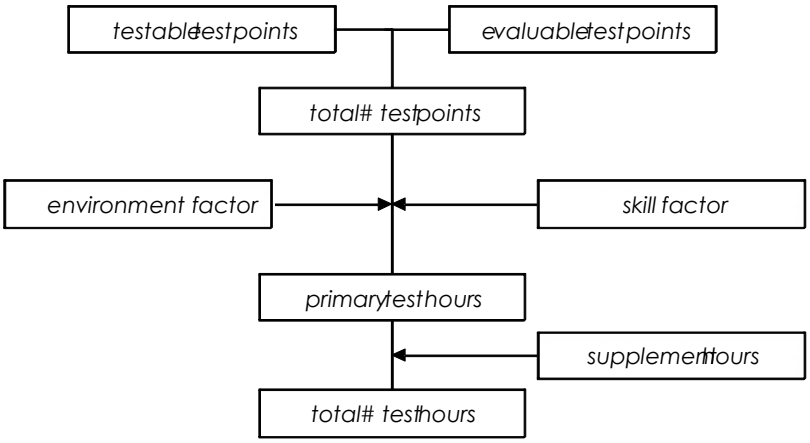


Figure 71. Schematic representation of test point analysis.

Based on the number of function points per function, the function-dependent factors (complexity, impact, uniformity, user importance and intensity of use), and the quality requirements and/or test strategy relating to the quality characteristics that must be tested, the number of test points that is necessary to test the testable quality characteristics is established per function (testable means that an opinion can be achieved on a specific quality characteristic by executing programs). Adding these test points over the functions results in the number of testable test points.

Based on the total number of function points of the information system and the quality requirements and/or test strategy relating to the quality characteristics that need to be evaluated, the number of test points that is necessary to evaluate those quality characteristics is established (evaluation: assessment of interim products without running software). This results in the number of evaluation test points.
The total number of test points is realized by adding the evaluable test points to the testable test points.

The primary test hours are then calculated by multiplying the total number of test points by the calculated environment factor and the applicable skill factor. The number of primary test hours represents the time necessary to execute the primary test activities. In other words, the time that is necessary to execute the test activities for the phases Preparation, Specification, Execution and Completion of the TMap life cycle.

The number of hours that is necessary to execute secondary test activities from the Control and Setting up and maintaining infrastructure phases (additional hours) is calculated as a percentage of the primary test hours.

Finally, the total number of test hours is obtained by adding the number of additional hours to the number of primary test hours. The total number of test hours is an estimate for all TMap test activities, with the exception of creating the test plan (Planning phase).

Principles

The following principles apply to test point analysis:

- Test point analysis is limited to the quality characteristics that are 'measurable'. Being measurable means that a test technique is available for the relevant quality characteristic. Moreover, sufficient practical experience must be available in relation to this test technique in terms of the relevant quality characteristic to make concrete statements about the required test effort.
- Not all possible quality characteristics that may be present are included in the current version of test point analysis. Reasons for this vary – there may be no concrete test technique available (yet), or there may be insufficient practical experience with a test technique and therefore insufficient reliable metrics available. Any subsequent version of test point analysis may include more quality characteristics.
- In principle, test point analysis is not linked to a person. In other words, different persons executing a test point analysis on the same information system should, in principle, create the same estimate. This is achieved by letting the client determine all factors that cannot be classified objectively and using a uniform classification system for all factors that can.
- Test point analysis can be performed if a function point count according to IFPUG [IFPUG, 1994] is available; gross function points are used as the starting point.
- Test point analysis does not consider subject matter knowledge as a factor that influences the required quantity of test effort in test point analysis. However, it is of course important that the test team has a certain level of subject matter knowledge. This knowledge is a precondition that must be complied with while creating the test plan.
- Test point analysis assumes one complete retest on average when determining the estimate. This average is a weighted average based on the size of the functions, expressed as test points.

Tip

From COSMIC full function points (CFFP) to function points (FP)

To estimate the project size, the COSMIC[9] Full Function Points (CFFP) approach is used more and more often in addition to the Function Point Analysis (FPA) approach [Abran, 2003]. FPA was created in a period in which only a mainframe environment existed and moreover relies heavily on the relationship between functionality and the data model. However, CFFP also takes account of other architectures, like client server and multi tier, *and* development methods like objected oriented, component based, and RAD.
The following rule of thumb can be used to convert CFFPs to function points (FPs):

- if CFFP < 250 : FP = CFFP
- if 250 ≤ CFFP ≤ 1000 : FP = CFFP / 1.2
- if CFFP > 1000 : FP = CFFP / 1.5

TPA, the technique in detail

4.9.7.3 Input and starting conditions

To perform a test point analysis, one must have a functional design. The functional design must include detailed process descriptions and a logical data model, preferably including a CRUD matrix. Moreover a function point count must have been executed according to IFPUG. These function point methods can be used as input for TPA. It is important to use only one of these function point methods when determining the skill factor, not multiple

[9] COSMIC: COmmon Software Measurement International Consortium

methods combined. In a function point count, the number of gross function points is taken as the starting point. Which function point method is used is not important when determining the test points. It will, however, have an impact on the skill factor.
The following modifications must be made to the function point count for TPA:

- The function points of the (logical) data collections distinguished in the function point count must be allocated to the function(s) that handle(s) the input of the relevant (logical) collection.
- The function points of the interface data collections distinguished in the function point count must be allocated to the function (or possibly functions) that use(s) the relevant interface data collection.
- For FPA functions in the clone class, the number of function points that applies to the original FPA function is used. A clone is an FPA function that has already been specified and/or realized in another, or the same, user function in the project.
- For FPA functions in the dummy class, the number of function points is determined if possible. Else this FPA function is given the qualification average complexity and the corresponding number of function points. A dummy is an FPA function if the functionality does not have to be specified and/or realized, but is already available because it was specified/realized outside the project.

Tip

Estimating guideline for counting function points
If no function point count is available and you wish to make one (for TPA), the following guideline can be used to determine the time required to count the function points: Determine the number of TOSMs using one of the methods described in section 11.3 and divide it by 400. The outcome represents an estimate of the number of days necessary to count the function points.
Note: as a rule, 350 to 400 function points can be counted in a day.

Calculation example (1)

Number of function points (FPf)
An information system has two user functions and one internal logical data collection:
Registration (11 function points), with as underlying FPA functions:

Entry	3 function points
Editing	4 function points
Deleting	4 function points

Processing (12 function points), with as underlying FPA functions:

Overview 1	5 function points
Overview 2	7 function points

The internal logical data collection 'data' has 7 function points and is allocated to the entry function in the context of test point analysis.

FP_f Registration 18 function points
FP_f Processing 12 function points

(**FP_f** = function points per function)

4.9.7.4 Testable test points

The number of testable test points is the sum of the number of test points per function in relation to testable quality characteristics. The number of test points is based on two types of factors:

- function-dependent factors (D_f)
- factor representing the testable quality characteristics (Q_d)

The FPA function is used as a unit of function. When determining the user importance and intensity of use, the focus is on the user function as a communication resource. The importance the users attach to the user function also applies to all of the underlying FPA functions.

Function-dependent factors

The function-dependent factors are described below, including the associated weights. Only one of the three described values can be selected (i.e. intermediate values are not allowed). If too little information is available to classify a certain factor, it must be given the nominal value (in bold print in this section).

User importance

User importance is defined as the relative importance the user attaches to a specific function in relation to the other functions in the system. As a rule of thumb, around 25% of the functions must be in the category "high", 50% the category "neutral", and 25% in the category "low".
User importance is allocated to the functionality as experienced by the user. This means allocation of the user importance to the user function. Of course, the user importance of a function must be determined in consultation with the client and other representatives of the user organization.

Weight

3 low: the relative importance of the specific function in relation to the other functions is low
6 Neutral: the relative importance of the specific function in relation to the other functions is neutral
12 high: the relative importance of the specific function in relation to the other functions is high.

Intensity of use

Intensity of use is defined as the frequency at which a certain function is used by the user and the size of the user group that uses that function.
As with user importance, intensity of use is allocated to functionality as experienced by users, i.e. the user functions.

Weight

2 low: the function is executed by the user organization just a few times per day or per week
4 neutral: the function is executed by the user organization many times per day
8 high: the function is executed continuously (at least 8 hours per day).

System impact

System impact is the level at which a mutation that occurs in the relevant function has an impact on the system. The level of impact is determined by assessing the logical data

collections (LDCs) to which the function can make mutations, as well as the number of other functions (within the system boundaries) that access those LDCs.

The impact is assessed using a matrix that shows the number of LDCs mutated by the function on the vertical axis, and the number of other functions accessing these LDCs on the horizontal axis. A function counts several times in terms of impact when it accesses multiple LDCs that are all maintained by the function in question.

No. of LDCs	Functions		
	1	2 - 5	> 5
1	L	L	M
2 - 5	L	M	H
> 5	M	H	H

Explanation: L = Low impact, M = Medium impact, H = High impact.

If a function does not mutate any LDCs, it has a low impact. A CRUD matrix is very useful when determining the system impact.

Weight

2 the function has a low impact
4 the function has a medium impact
8 the function has a high impact.

Complexity

The complexity of a function is assessed on the basis of its algorithm. The global structure of the algorithm may be described by means of pseudo code, Nassi-Shneidermann or regular text. The level of complexity of the function is determined by the number of conditions in the algorithm of that function. When counting the number of conditions, only the processing algorithm must be taken into account. Conditions resulting from database checks, such as validations by domain or physical presence, are not included since they are already incorporated implicitly in the function point count.

As such the complexity can be determined simply by counting the number of conditions. Composite conditions, such as IF a AND b THEN count double for complexity. This is because two IF statements would be needed without the AND statement. Likewise, a CASE statement with n cases counts for n-1 conditions, because the replacement of the CASE statement by successive IF statements would result in n-1 conditions. In summary: count the conditions, not the operators.

Weight

3 a maximum of 5 conditions are present in the function
6 6 to 11 conditions are present in the function
12 more than 11 conditions are present in the function.

Uniformity

In three types of situation, a function counts for only 60%:

- A nearly unique function occurring a second time – in this case, the test specifications that are to be defined can be largely reused.
- Clones – in this case, too, the test specifications that are to be defined can be reused.
- Dummy functions – but only if reusable test specifications for the dummy exist.

The uniformity factor is given the value 0.6 if one of the above conditions is met, otherwise it is given the value 1.

In an information system, there can be functions that have a certain level of uniformity in the context of testing, but are marked as unique in the function point analysis. In the function point analysis, being unique means:

- A unique combination of data collections in relation to the other input functions.
- Not a unique combination of data collections, but another logical processing method (e.g. updating a data collection another way).

In addition, there are functions in an information system that are said to be fully uniform in the context of function point analysis and are therefore not allocated any function points, but must be counted in the testing because they do require testing. These are the clones and dummies.

Calculation method

The factor (D_f) is determined by establishing the sum of the values of the first four function-dependent variables (user importance, intensity of use, system impact and complexity) and dividing it by 20 (the nominal value). The result of this calculation must then be multiplied by the value of the uniformity factor. The D_f factor is determined per function.

D_f = ((Ui + Iu + Si + C) / 20) * U

D_f = weight factor of the function-dependent factors
Ui = user importance
Iu = intensity of use
Si = system impact
C = complexity
U = uniformity

Standard functions

If functions for error messages, help screens and/or menu structure are present in the function point count – which often is the case – they must be valued as follows:

Function	FPs	Ui	Ui	Si	C	U	D_f
Error messages	4	6	8	4	3	1	1.05
Help screens	4	6	8	4	3	1	1.05
Menu structure	4	6	8	4	3	1	1.05

Calculation example (2)

Determining the function-dependent variables (D_f)

	Registration	Processing
User importance	6	12
Intensity of use	8	2
System impact	2	2
Complexity	3	6
Uniformity	1	1
D_f =	19/20 * 1 = 0.95	22/20 * 1 = 1.10

(In this example, it is assumed that the valuation of the factors system impact and complexity are identical for the FPA functions in a user function.)

Testable quality characteristics

Below, we describe how the requirements specified for the testable quality characteristics are incorporated into the test point analysis. In relation to the testable quality characteristics, TPA distinguishes between quality characteristics that can be measured explicitly and/or implicitly.
The following can be measured explicitly:

- functionality
- security
- effectivity/suitability
- performance
- portability.

The weight of the quality requirements must be valuated for each quality characteristics in the context of the test to be executed, by means of a score, possibly by sub-system.

Weight

0 not important – not measured
3 low quality requirements – attention must be devoted to it in the test
4 regular quality requirements – usually applicable if the information system relates to a support process
5 high quality requirements – usually applicable if the information system relates to a primary process
6 extremely high quality requirements.

The quality characteristics that are measured explicit have the following weight factors:

Functionality	0.75
Security	0.05
Effectivity	0.10
Performance	0.05
Portability	0.05

Which relevant quality characteristics (distinguished in the test strategy) will be tested implicit must be determined. A statement about these quality characteristics can be made by collecting statistics during test execution. E.g. performance can be measured explicitly, by means of a real-life test, or implicitly, by collecting statistics.

The quality characteristics to be measured implicit must be specified. The number of quality characteristics can then be determined. The weight is 0.02 per characteristic for Q_d. In principle, every quality characteristic can be tested implicit.

Calculation method (Q_d)

The score given to each explicit measurable quality characteristic is divided by four (the nominal value) and then multiplied by its weight factor. The sum of the figures obtained this way is calculated.

If certain quality characteristics were earmarked for implicit testing, the associated weight (0.02 per characteristic) must be added to the above sum. The figure obtained this way is the Q_d factor. Usually, the Q_d factor is established for the total system once. However, if the strategy differs per sub-system, the Q_d factor must be determined per sub-system.

Calculation example (3)

Determining the testable quality characteristics (Q_d)

Functionality	5	(5/4) * 0.75 = 0.94
Security	4	(4/4) * 0.05 = 0.05
Effectivity	0	(0/4) * 0.10 = 0
Performance	0	(0/4) * 0.05 = 0
Portability	0	(0/4) * 0.05 = 0

The following are measured implicit:

Performance	= 0.02
Economy	= 0.02
Maintainability	= 0.02

Q_d = 0.94 + 0.05 + (3 * 0.02) = 1.05

Formula for testable test points

The number of testable test points is a sum of the number of test points per function. The number of test points per function can be established by entering what is now known in the formula below:

$$\mathbf{TP_f = FP_f * D_f * Q_d}$$

TP_f = the number of test points per function
FP_f = the number of function points per function
D_f = weight factor of the function-dependent factors
Q_d = weight factor of the testable quality characteristics

Calculation example (4)

Calculation of total number of testable test points ($\sum TP_f$)

	FP_f *	**D_f ***	**Q_d**	**=**	**TP_f**
Registration	18	0.95	1.05	=	18
Processing	12	1.10	1.05	=	14 +
Total number of testable test points					32

4.9.7.5 Evaluable test points

The number of evaluable test points naturally depends on the quality characteristics that require evaluations (the Q_s factor), but also on the total number of function points of the system. An evaluation of a large-scale information system simply takes more time than one of a simple information system.

For the relevant quality characteristics, it must be determined whether or not they will be evaluated. A statement about these quality characteristics is achieved by means of a checklist. In principle, all quality characteristics can be evaluated with the aid of checklists. E.g. security can be measured either by testing explicitly, with the aid of a semantic test, or by evaluating the security measures on the basis of a checklist.

Calculation method (Q_S)

If a quality characteristic is evaluated, the factor Q_s will have a value of 16. For each subsequent quality characteristic to be included in the evaluation, another value of 16 is added to the Q_s factor rating.

Calculation example (5)

Calculation of evaluable test points (Q_s)

The following quality characteristics are evaluated (using a checklist):
Continuity = 16
Q_s = 16

4.9.7.6 Total number of test points

The number of test points of the total system can be established by entering what is now known in the formula below:

$$\mathbf{TP = \sum TP_f + ((FP * Q_s) / 500)}$$

TP = the number of test points of the total system
$\sum TP_f$ = the sum of the number of test points per function (testable test points)
FP = number of function points of the total system (minimal value 500)
Q_s = weight factor of quality characteristics to be evaluated

Calculation example (6)

Calculation of total number of test points (TP)

TP = 32 + ((500 * 16) / 500) = 48

4.9.7.7 Primary test hours

The formula in the section above results in the total number of test points. This is the measure for the scope of the primary test activities. These primary test points are multiplied by the skill factor and the environment factor to obtain the primary test hours. This represents the time that is necessary to execute the test activities for the Preparation, Specification, Execution and Completion phases of the TMap model.

Skill factor

The skill factor indicates how many hours of testing are required per test point. The higher skill factor, the greater the number of hours of testing.

The productivity with which the test object is tested on the basis of the test strategy depends primarily on the knowledge and skills of those executing the tests. It is also relevant to know if people are testing part-time or full-time. Testing users that are deployed for test work only part of the workday, have a lot of switch moments between their day-to-day work and the test work, which often results in reduced productivity.

In practice, the following basic figures are used per test point:
- 1-2 hours for a tester, depending on knowledge and skills
- 2-4 hours for a user, depending on experience.

The skill factor naturally varies per organization and within that even per department/person. A factor can be obtained by analyzing completed test projects. To make such an analysis, one must have access to experience figures for the test projects already realized.

Calculation example (7)

Skill factor

For the relevant organization, a skill factor of 1.2 applies.
S = 1.2

Environment factor

The number of required test hours per test point is influenced not only by the skill factor, but by the environment factor as well.

A number of environment variables are used to calculate this. The environment variables are described below, including the associated weights. Again, only one of the available values may be selected. If too little information is available to classify a certain variable, it must be given the nominal value (in bold print).

Test tools

The test tools factor involves the level to which the primary text activities are supported by automated test tools. Test tools can contribute to executing part of the test activities automatically and therefore faster. Their availability does not guarantee that, however - it is about their effective use.

Weight

1 the test uses support tools for test specification, and a tool is used for record & playback
2 test execution uses support tools for test specification, or a tool with record & playback options is used
4 no test tools are available.

Previous test

For this factor the quality of the test executed earlier is important. When estimating an acceptance test this is the system test, when estimating a system test, the development test. The quality of the previous test is a co-determinant for the quantity of functionality that may be tested at a more limited level as well as for the lead time of the test execution. When the previous test is of a higher quality, fewer progress-hindering defects will occur.

Weight

2 a test plan is available for the previous test, and the test team also has insight into the concrete test cases and test results (test coverage)
4 a test plan is available for the previous test
8 no test plan is available for the previous test.

Test basis

The test basis is awarded a factor representing the quality of the (system) documentation on which the test for execution mu st be based. The quality of the test basis has an impact in particular on the required time for the Preparation and Specification phases.

Weight

3 standards and templates are used to create the system documentation. The documentation is also subject to inspections
6 standards and templates are used to create the documentation
12 no standards and templates are used to create the system documentation.

Development environment

The environment in which the information system is realized. Of particular interest here is to what extent the development environment prevents errors and/or enforces certain things. If certain errors can no longer be made, clearly they do not require testing.

Weight

2 the development environment contains a large number of facilities that prevent errors being made for example by executing semantic and syntactic checks and by taking over the parameters
4 the development environment contains a limited number of facilities that prevent errors being made for example by executing a syntactic check and by taking over the parameters
8 the development environment contains no facilities that prevent errors being made.

Test environment

The extent to which the physical test environment in which the test is executed has proven itself. If an often used test environment is used, fewer disturbances and defects will occur during the Execution phase.

Weight

1 the environment has already been used several times to execute a test
2 a new environment has been set up for the test in question, the organization has ample experience with similar environments
4 a new environment has been set up for the test in question that can be characterized as experimental for the organization.

Testware

The level to which existing testware can be used during the test to be executed. The availability of effective testware has a particular impact on the time required for the Specification phase.

Weight

1 a usable general central starting situation (tables etc) is available, as well as specified test cases for the test to be executed
2 a usable general central starting situation (tables etc) is available
4 no usable testware is available.

Calculation method

The environment factor (E) is determined by establishing the sum of the values of the environment variables (test tools, previous test, test basis, development environment, test environment, and testware) and dividing it by 21 (the nominal value). The environment factor E can be established for the total system once, but also per sub-system if necessary.

Calculation example (8)

Environment factor (E)

The various environment variables were given the score below:

Test tools	4 (no test tools)
Previous test	4 (a test plan is available of the previous test)
Test basis	3 (documentation templates and inspections)
Development environment	4 (Oracle in combination with COBOL)
Test environment	1 (tested environment)
Testware	4 (no usable testware available)

E = 20/21 = 0.95

Formula for primary test hours

The number of primary test hours is obtained by multiplying the number of test points by the skill and environment factors:

PT = TP * S * E

PT = the total number of primary test hours
TP = the number of test points of the total system
S = skill factor
E = environment factor

Calculation example (9)

Calculation of primary test hours (PT)

PT = 48 * 1.2 * 0.95 = 54.72 (55 hours)

4.9.7.8 The total number of test hours

Since every test process involves secondary activities from the Control phase and the Setting up and maintaining infrastructure phase, a supplement must be added to the primary test hours for this. This will eventually result in the total number of test hours. The number of supplemental hours is calculated as a percentage of the primary test hours.

The supplemental percentage is often determined by a test manager on the basis of experience or using historical data. Some organizations use a fixed percentage. The percentage is nearly always in the range of 5 to 20%.

If no experience, historical data or fixed percentages are available, a supplemental percentage can be estimated in the following way. A standard (nominal) supplemental percentage of 12% is used as the starting point. We must then look at factors that may increase or reduce the percentage. Examples of such factors are:

- Team size
- Management tools
- Permanent test organization.

These factors are explained below. Since there is a great variety of test projects, we have not used seemingly certain absolute percentage figures to determine the impact of these factors on the percentage, but have chosen to indicate whether the impact will increase or reduce the percentage.

Team size

The team size represents the number of members in the test team (including the test manager and a test administrator, if any). A big team usually results in greater overhead and therefore a higher supplemental percentage. However, a small test team results in a reduced percentage:

- Reduction
  Test team consists of maximum 4 persons.
- Neutral
  Test team consists of 5 - 9 persons.
- Increase
  Test team consists of at least 10 persons.

Management tools

For management tools, it is considered to what extent automated tools are used during the test activities for Control and Setting up and maintaining infrastructure. Examples of these tools are an automated:

- planning system
- progress monitoring system
- defect administration system
- testware management system.

If little use is made of automated tools, certain activities will have to be done manually. This increases the supplement percentage. If intensive use is made of automated tools, this will reduce the percentage:

- Reduction
  At least 3 automated tools are used.
- Neutral
  1- 2 automated tools are used.
- Increase
  No automated tools are used.

Permanent test organization

There are many kinds of permanent test organization (section 8.3). If an organization has one of these permanent test organizations, lead time reduction, cost savings and/or quality improvement are often realized in a test process that uses it:

- Reduction
  Test team uses the services of a permanent test organization.
- Neutral
  Test team does not use the services of a permanent test organization.

Calculation example (10)

Determining supplement for Control and Setting up and maintaining infrastructure (C and S&MI)

Historical data show the supplement percentage for such test projects to fluctuate around 15%. The test manager decides to use this percentage.

Supplement percentage C and S&MI = 15%

Calculation method

The supplemental percentage is used to calculate the supplement (in hours) on the basis of the number of primary test hours. The total number of test hours is then obtained by

adding the supplement calculated for Control and Setting up and maintaining infrastructure to the total number of primary test hours.

Calculation example (11)

Calculation of total number of test hours
Primary test hours 55
Supplement M and S&MI 55 * 0.15 = 8.25 (rounded down: 8)

Total number of hours 55 + 8 = 63

Figure 72 shows the TPA calculation example as a whole.

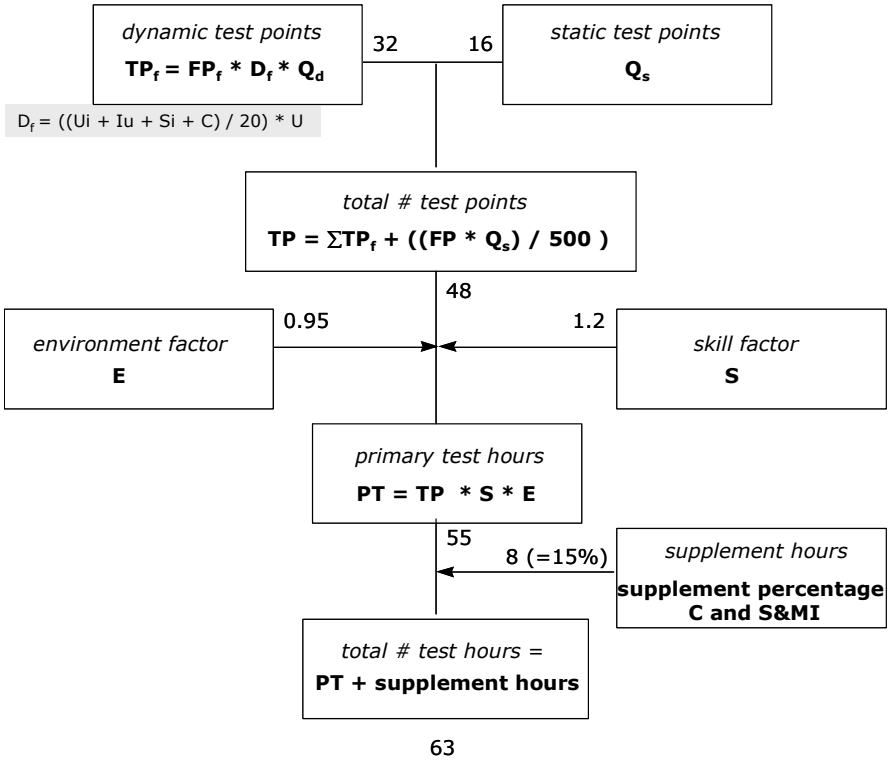


Figure 72. Schematic representation of calculation example.

4.9.7.9 Distribution over the phases

When using TMap, the test process is split up into seven phases, and many clients will be interested in the estimate per phase in addition to the estimate for the entire test process.

TPA gives an estimate for the entire test process, with the exception of test plan creation (Planning phase).

In principle, for the phases Control and Setting up and maintaining infrastructure, the number of hours is estimated that was calculated on the basis of the number of primary test hours using the supplement percentage (supplement hours). These supplement hours must be divided between the two phases.

The primary test hours are divided over the other phases (Preparation, Specification, Execution, and Completion). The distribution of the primary test hours over the phases may naturally vary per organization, and even within one organization. A distribution applicable to the organization can be obtained by analyzing completed test projects. To make such an analysis, one must have access to experience figures for the test projects already realized.

Distribution of primary test hours

Practical experience with test point analysis in combination with TMap yields the following distribution of the test effort over the various phases:

Preparation	10%
Specification	40%
Execution	45%
Completion	5%

4.9.7.10 TPA at an early stage

Often, a project estimate for testing must be made at an early stage. In this case, it is not possible to establish factors like complexity, impact and so on, because no detailed functional specifications are available. However, there are approaches that can often be used to perform a rough test point analysis. By using one of the approaches below, the total number of (gross) function points can be estimated:

- on the basis of very rough specifications, perform a so-called rough function point analysis
- determine the number of function points by determining the number of TOSMs.

One function is then defined for the purpose of a rough test point analysis. This function has the size of the total number of defined (gross) function points. In principle, all function-dependent factors (user importance, intensity of use, complexity, impact and uniformity) are given the neutral value, so that Df = 1. A test point analysis can then be made as described in the previous sections. Usually assumptions will have to be made when determining the environment factor. When presenting the test estimate, it is important to describe these assumptions clearly.

4.10 Metrics

4.10.1 Introduction

Quite often, test managers are expected to answer such questions as:

- Why does testing take so long?
- Why has the test process not been completed yet?
- How many defects can I still expect during production?
- How many re-tests are still required?
- When can testing be stopped?
- When will the test team start the execution of the test?
- Tell me what exactly you are up to!
- What is the quality of the system that you have tested?
- When can I start production?
- How can it be that the previous test project was much faster?
- What did you actually test?
- How many defects have been found and what is their status?

Answering these types of questions with well-founded, factually based answers is not easy. Most questions can be answered with reference to periodic reports. Such reports can only be created on the basis of correctly recorded relevant data, which is converted into information and then used to answer the above-mentioned questions.

Metrics on the quality of the test object and the progress of the test process are of great importance to the test process. They are used to manage the test process, to substantiate test advice and also to compare systems or test processes with each other. Metrics are important for improving the test process, in assessing the consequences of particular improvement measures by comparing data before and after the measures were adopted.

To summaries, a test manager should record a number of items in order to be able to pass well-founded judgment on the quality of the object under test as well as on the quality of the test process itself. The following sections describe a structured approach for arriving at a set of test metrics.

4.10.2 GQM method in six steps

There are various ways of arriving at a particular set of metrics. The most common is the Goal-Question-Metric (GQM) method [Basili, 1994]). This is a top-down method in which one or more goals are formulated. For example: what information should I collect in order to answer those questions posed in the introduction? These goals include questions that constitute the basis for the metrics. The collected metrics should provide the answers to those questions, and the answers will indicate among other things whether the goal has been achieved or not. The summary of the GQM method described below focuses in particular on the test aspect. The GQM process is described in six steps. This is a concise description that includes only those items that are relevant to the test manager. For a more detailed description, please refer to the aforementioned GQM literature.

4.10.2.1 Step 1: Defining the goals

Measuring purely for the sake of measuring is pointless. Clear and realistic goals should be set beforehand. We distinguish two types of goals:

- Knowledge goals (knowing where we are now). These goals are expressed in words such as evaluate, predict, or monitor. For example, "Evaluate how many hours are actually spent on re-testing" or "Monitor the test coverage". The goal here is to gain insight.
- Improvement goals (where do we want to go). These goals are expressed in words such as increase, decrease, improve, or achieve. Setting such goals suggests that we know there are shortcomings in the present test process or the present environment and that we want to improve these.

An example of such an improvement goal is obtaining a 20% saving on the number of testing hours at a constant test coverage within a period of 18 months.
In order to ascertain this, the following two knowledge goals should be aimed at:

- Insight into the total number of testing hours per project.
- Insight into the achieved test coverage per project.

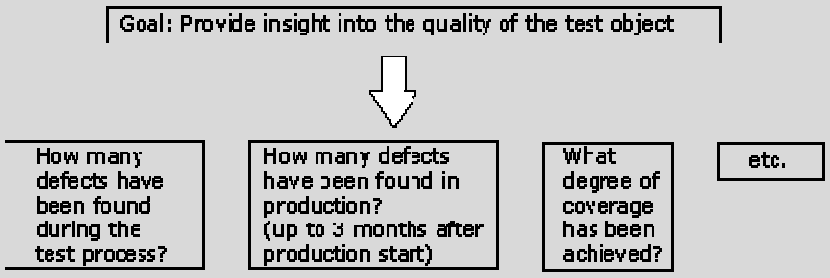
It is important to investigate whether the goals and the (test) maturity of the organization match. It is pointless to aim at achieving a certain test coverage if the necessary resources (knowledge, time and tools) are not available.
Example: Knowing where we are now.

Example

4.10.2.2 Step 2: Asking questions per goal

For each goal, several questions have to be asked. The questions are formulated in such a way that they act as a specification of a metric. It can also be asked, for each question, who is responsible for the test metrics supplied. From the above goal, various questions can be derived. We will limit the number of questions in this example to three.

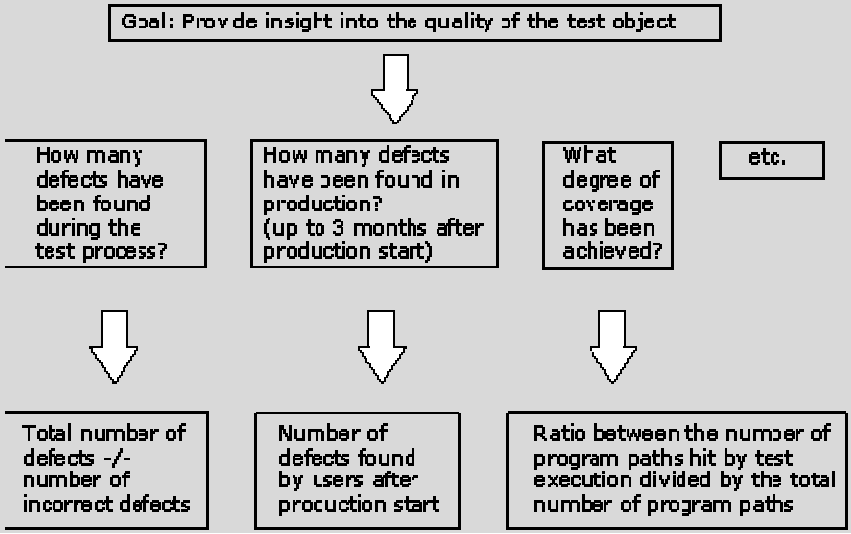
Example



4.10.2.3 Step 3: From questions to metrics

The relevant metrics are derived from the questions, and form the full set of metrics, gathered during the test process.

Example



By asking the right questions, we arrive at the correct set of metrics for a certain goal. It is important to define and specify each metric correctly. For example, what exactly is a defect?

4.10.2.4 Step 4: Data collection and analysis

During the test process a variety of data is collected. One way of keeping things simple is to use forms/templates (if possible in electronic form). The data should be complete and easy to interpret. In the design of these forms, attention should be paid to the following points:

- Which metrics are collected on the same form.
- Validation: how easy is it to check whether the data is complete and correct.
- Traceability: forms supplied with the date, project ID, configuration management data, data collector, etc. Take into consideration that it is sometimes necessary to preserve this data for a long time.
- Possibility of electronic processing.

As soon as the data are collected, it should be analyzed. At this point it is still possible to make corrections. Waiting too long decreases the chance of restoring the data. Bear in mind possibilities, for example, of booking time with the incorrect activity code.

4.10.2.5 Step 5: Presentation and distribution of the measurement data

The collected measurements are used both in the test reports on the quality of the product under test and in those on the test process. Proper feedback is also of importance to the motivation of those involved and the validation of the measured data.

4.10.2.6 Step 6: Relating the measurement data to the questions and goals

This last step is used to investigate to what extent the indicators (answers to the questions) offer sufficient insight into the matter of whether the goals have been achieved. This situation may be the starting point for a new GQM cycle. In this way, we are continually improving the test process.

4.10.3 Hints and tips

When metrics are being collected, the test manager should take the following issues into account:

- Start with a limited set of metrics and build it up slowly.
- Keep the metrics simple. The definition should appeal to the intuition of those involved. For example, try to avoid the use of a variety of formulas. The more complicated the formulas, the more difficult they are to interpret.
- Choose metrics that are relatively simple to collect and easily accepted. The more difficult it is to collect data, the greater the chance that it will not be accepted.
- Collect data electronically as much as possible. This is the quickest way of data collection and avoids the introduction of manual errors into the data set.
- Keep an eye on the motivation of the testers to record accurately. In the case of time registration, for example, it sometimes happens that incorrect (read: not yet fully booked) codes are used.
- Avoid complicated statistical techniques and models during presentations. Allow the type of presentation to depend on the data presented (tables, diagrams, pie charts, etc.).
- Provide feedback to the testers as quickly as possible. Show them what you do with the information.

4.10.4 Practical starting set of test metrics

Below is an indication of what test managers embarking on a "metrics programme" should start with. The metrics set described is a starting set that can be used in practice with little cost and effort. Section "Reporting" lists a number of more specific test statistics and progress reports.

- Registration of hours, using activity codes. Register the following for each tester: date, project, TMap phase, activity and number of hours. A "Comments" field is recommended, making it possible to check whether the data has been entered correctly. Registering the hours in this way enables you to obtain insight into the time spent on each TMap phase (see figure 73). It also enables the client to check the progress of the test process. It is advisable to compile this type of timesheet on a weekly basis for projects that last up to three or four months. For projects that last longer than half a year, this can be done on a fortnightly basis. For projects that last longer than a year, it is best to report on a monthly basis.
- Collect data about the test deliverables (test plans, test scripts, etc.), the test basis and test object. Record the following: document name, delivery date, TMap phase upon delivery, version and a characteristic that says something about the quantity. This may be the number of test cases for the test scripts, or the number of pages for the other documents. For the test basis, the number of user requirements can be taken as a quantity characteristic.
- Report on the progress of the defects. Section 4.7 "Defects management" describes how a defect administration can be set up. An example of this type of reporting is shown in figure 74:

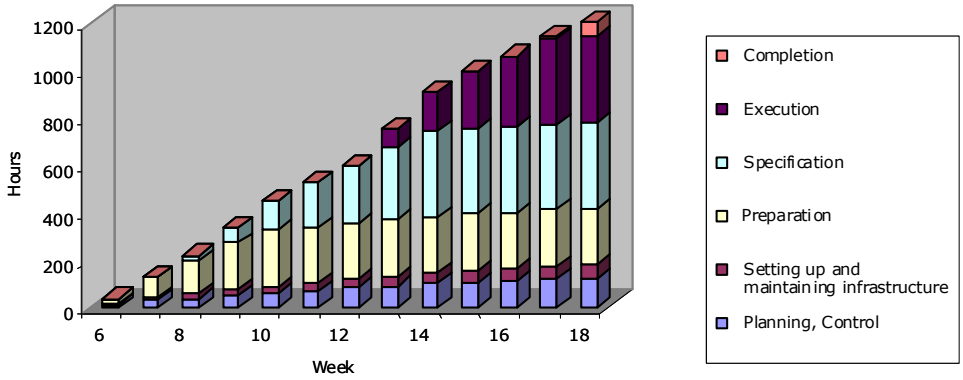


Figure 73. Example of hours spent on test process, per TMap phase.

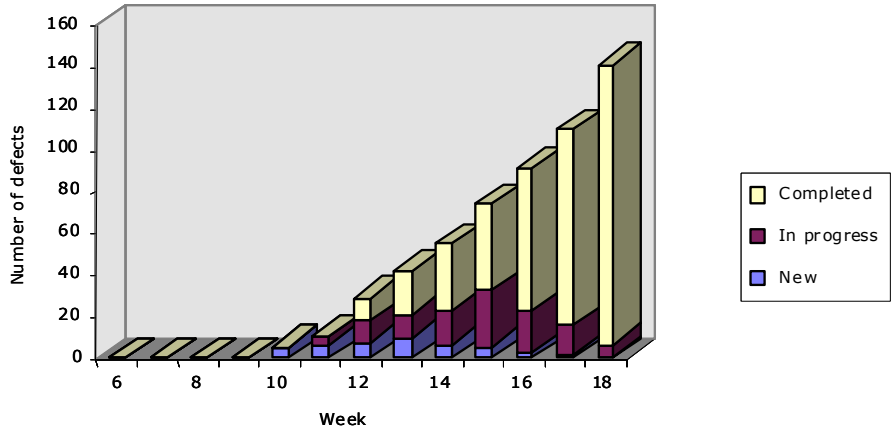


Figure 74. Example of a progress overview of defects.

These elementary metrics (hours, documents and defects) can be used to assess the productivity of the test process. Note that this productivity should be seen in relation to the required effort and size of the test project. Example: in the first ten hours of testing we may find more defects per hour than in 400 hours of further testing, simply because the first defects are found more quickly than the last ones.

The following metrics regarding productivity can be derived from this elementary set:
- number of defects per hour (and per hour of test execution)
- number of test cases carried out per hour
- number of specified test scripts per hour (and per hour of test specification)
- number of defects per test script
- ratio of hours spent over the TMap phases.

If the number of function points or the number of 'kilo lines of code' (KLOC) of the object under test is known, the following numbers can be calculated:
- number of test hours per function point (or KLOC)
- number of defects per function point (or KLOC)
- number of test cases per function point (or KLOC).

For the test basis we can establish the following metrics:
- number of test hours per page of the test basis
- number of defects per page of the test basis
- number of test cases per page of the test basis
- number of pages of the test basis per function point.

When it is known how many defects occur in production during the first three months, the following metric can be determined:
- Defect-detection effectiveness of a test level: number of found defects in a test level divided by the total number of defects present. This metric is also called the Defect Detection Percentage (DDP). In calculating the DDP, the following assumptions are applied:
  - all the defects are included in the calculation
  - the weighed severity of the defects is not included in the calculation
  - after the first three months of the system being in production, barely any defects are present in the system.

  The DDP can be calculated both per test level and overall. The DDP per test level is calculated by dividing the number of found defects from the relevant test level by the sum of this number of found defects and the number of found defects from the subsequent test level(s) and/or the first three months of production. The overall DDP is calculated by dividing the total number of found defects (from all the test levels together) by the sum of this number of found defects and the found defects from the first three months of production.

Example

DDP calculations

Test level	Found defects
System test (ST)	100
Acceptance test (AT)	60
3 months of production	40

DDP ST (after the AT is carried out)	: (100	/ 100+60	) = 63%
DDP ST (after 3 months of production)	: (100	/ 100+60+40	) = 50%
DDP AT (after 3 months of production)	: (60	/ 60+40	) = 60%
DDP overall	: (100+60	/ 100+60+40	) = 80%

Some causes of a high or low DDP may be:

- High DDP
  - the tests have been carried out very accurately
  - the system has not yet been used much
  - the subsequent test level was not carried out accurately.
- Low DDP
  - the tests have not been carried out accurately
  - the test basis was not right, consequently nor were the tests derived from it
  - the quality of the test object was wrong (containing too many defects to be found during the time available)
  - the testing time has been shortened.

By recording the above-mentioned metrics, supplemented here and there with particular items, we arrive at the following list of metrics.

4.10.5 Metrics list

In the following (non-exhaustive) list of metrics, a number of commonly used metrics are mentioned, which can be used as indicators for pronouncing on the quality of the object under test or for measuring the quality of the test process and comparing against the organization's standard. All the indicators can of course also be used in the report to the client:

- *Number of defects found*
  The ratio between the number of defects found and the size of the system per unit of testing time.
- *Executed instructions*
  Ratio between the number of tested program instructions and the total number of program instructions. Tools that can produce such metrics are available.
- *Number of tests*
  Ratio between the number of tests and the size of the system (for example expressed in function points). This indicates how many tests are necessary in order to test a part.
- *Number of tested paths*
  Ratio between the tested and the total number of logical paths present.
- *Number of defects during production*
  This gives an indication of the number of defects not found during the test process.
- *Defect detection effectiveness*
  The total number of defects found during testing, divided by the total number of defects – estimated partly on the basis of production data.
- *Test costs*
  Ratio between the test costs and the total development costs. A prior definition of the various costs is essential.
- *Cost per detected defect*
  Total test cost divided by the number of defects found.
- *Budget utilization*
  Ratio between the budget and the actual cost of testing.
- *Test efficiency*
  The number of required tests versus the number of defects found.
- *Degree of automation of testing*
  Ratio between the number of tests carried out manually and the number of tests carried out automatically.

- *Number of defects found (relative)*
  The ratio between the number of defects found and the size of the system (in function points or KLOC) per unit of testing time.
- *Defects as a result of modifications that are not tested*
  Defects because of modifications that are not tested, as a part of the total number of defects arising as a result of changes.
- *Defects after tested modifications*
  Defects because of modifications that are tested, as a part of the total number of defects arising as a result of changes.
- *Savings of the test*
  Indicates how much has been saved by carrying out the test. In other words, what would the losses have amounted to if the test had not been carried out?

4.11 Evaluation techniques

4.11.1 Introduction

The result of an evaluation depends heavily on the attitude of the evaluator(s). After all, an evaluation often assesses documents created by someone else. It usually results in a list of findings intended for the author of the document in question. Depending on how the findings are recorded or communicated, the author may feel 'attacked'. Chances are that this will result in a negative perception of the evaluation process. As such, it is important to realize that the author did not intend to write things down 'incorrectly' on purpose. It is also important to be aware that the evaluation process' final goal is to deliver the best possible end product together. Evaluation techniques are very well suited to improve the quality of products. This applies not only to the evaluated products themselves, but also to other products. For instance, the findings from the evaluation process may cause the development process to implement process improvements.

Research has shown, however, that evaluation processes – despite their proven value – are not always implemented or executed seamlessly. A few causes that a survey brought to light:

- 56% of the authors found it hard to disengage from their work
- 48% of the evaluators had not received the correct training
- 47% of the authors were afraid that the data would be used against them.

After a general description of evaluation, this section discusses three techniques in greater detail: inspections, reviews, and walkthroughs. The last subsection of this section contains an evaluation technique selection matrix that can be used to choose a technique.

"IEEE Std 1028-1997 Standard for Software Reviews" [IEEE, 1998] was used as a basis for this chapter, supplemented with experiences from practice.

4.11.2 Evaluation explained

4.11.2.1 Products

Various products are developed in the course of the system development process. These can either be intermediary products or end products. Depending on the selected method, these have a certain form, content and inter-relationship, on the basis of which they can be evaluated.

Definition Evaluation is assessing the products in the system development process without running software.

As far as intermediary products are concerned, evaluation does not have to limit itself to the development documents. Evaluation can occur at all levels of documentation:

- Functional/ technical design
- Requirements document
- Management plan
- Development plan
- Test plan
- Maintenance documentation
- User/Installation manual
- Software

- Release note
- Test design
- Test script
- Prototype
- Screen print
- Etc.

4.11.2.2 Position of evaluations

Beside the quality improvement described earlier, another important aspect of evaluation is that defects can be found much earlier in the (development) process than by testing (see figure 75). This is because evaluation assesses intermediary products and not, as testing does, the end products. And the earlier a defect is found, the more easily and economically it can be repaired [Boehm, 1981].

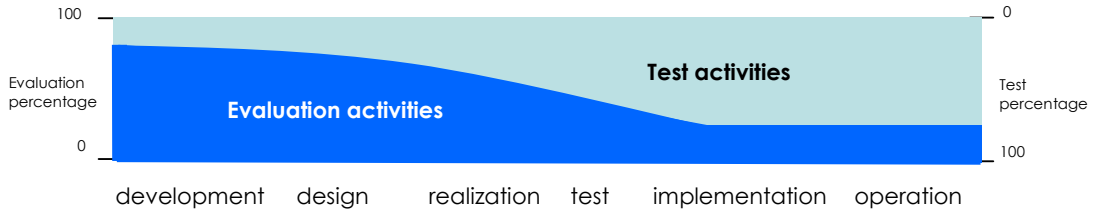


Figure 75. Evaluation and testing versus system development phasing.

Evaluation has demonstrated time and again to be the most efficient and effective way to eliminate defects from the intermediary products of a system (see the practical examples in section "Preparation phase"). Evaluation is also a process that is easy to set up because, for instance, no software has to be run and no environment has to be created. As such, there are adequate reasons to set up an evaluation process.

In practice, however, good intentions can become mired in practical execution problems: "Can you have a look at these six folders with the functional specifications? Please get them back to us the day after tomorrow, because that is when we start programming."

Formal evaluation techniques in the form of process descriptions and checklists, help get this under control. A formal evaluation technique is characterized by the fact that several persons evaluate as a team, defects are documented (see tip), and there are written procedures to execute the evaluation activities.

Tip

The evaluators detect many defects; often it is decided to enter these in a defect administration. At the end of the evaluation meeting, rubrics such as status, severity, and action have to be updated for the defects. These are labor and time-intensive activities that can be minimized as follows:

- Evaluators record their comments on an evaluation form.
- These comments are discussed during the evaluation meeting.
- At the end of the meeting, only the most important comments are registered in the defect administration as a defect. Please refer to section 4.7 "Defects management", for more information on the defect procedure.

An evaluation form contains the following aspects:

Per form

- Identification of the evaluator

    - name
    - role
- Identification of the product
    - document name
    - version number/date
- Evaluation process data
    - number of pages evaluated
    - evaluation time invested
- General impression of the product

Per comment
- Unique reference number
- Clear reference to the place in the product to which the comment relates (e.g. by specifying the chapter, section, page, line, requirement number)
- Description of the comment
- Importance of the comment (e.g. high, medium, low)
- Follow-up actions (to be filled out by the *author* with e.g. completed, partially completed, not completed)

Note: This form is useful for reviewing documents in particular. When reviewing software or for a walkthrough of a prototype, the aspects on the form need to be modified.

There are various evaluation technique intensities. This is important because not every intermediary product needs to be evaluated equally intensively. This is why an evaluation strategy is often included in the master test plan. Like a test strategy, an evaluation strategy is extremely important when aiming to deploy the effort in an optimal way, as well as a means of communication towards the client. While establishing the strategy, it is analyzed what has to be evaluated where and how often in order to achieve the optimal balance between the required quality and the amount of time and/or money that is required. Optimization aims to distribute the available resources correctly over the activities to be executed.

Structure of evaluation technique sections

After a few general tips, the subsections below describe the techniques according to the following structure:
- Introduction
  Description of the goal of the technique and the products to which the technique can be applied.
- Responsibilities
  Description of which roles are allocated to participants.
- Entry criteria
  Description of the necessary products and conditions that must be met before evaluation can begin.
- Procedures
  Description of:
    - How to organize an evaluation (planning)
    - The method to be used
    - The required preparation of the participants
    - How evaluation results are discussed and recorded
    - How rework is done and checked.
- Exit criteria
  Description of the deliverable documents and the conditions under which the product can exit the evaluation process.

Tips

When using an evaluation technique, practice has shown that there are several critical success factors:

- The author must be released from other activities to participate in the evaluation process and process the results.
- Authors must not be held accountable for the evaluation results.
- Evaluators must have attended a (short) training in the specific evaluation technique.
- Adequate (preparation) time must be available between submission of the products to the evaluators and the evaluation meeting (e.g. two weeks). If necessary, the products can be made available during a kick-off. See figure 76 for a possible planning of the evaluation programme.
- The minutes secretary must be experienced and adequately instructed. Making minutes of all defects, actions, decisions and recommendations of the evaluators is vital to the success of an inspection process. Sometimes the author takes on the role of minutes secretary, but the disadvantage is that the author may miss parts of the discussion (because he must divide his attention between writing and listening).
- The size of the intermediary product to be evaluated and the available preparation and meeting time must be tuned to each other.
- Make clear follow-up agreements. Agree when and in which version of the intermediary product the agreed changes must be implemented.
- Feedback from the author to the evaluators (appreciation for their contribution).

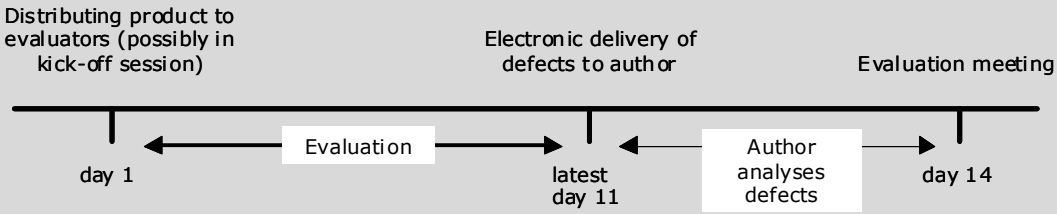


Figure 76. Possible planning of an evaluation programme.

The planning above does not keep account of any activities to be executed after the evaluation meeting. These might involve: modifying the product, re-evaluation, and final acceptance of the product. If relevant, such activities must be added to the planning. The activities 'modifying the product' and 're-evaluation' may be iterative.

Practical example

In an organization in which the time-to-market of various modifications to the information system had to be short, the modifications were implemented by means of a large number of short-term increments.

The testers were expected to review the designs (in the form of use cases). A lead time of two weeks for a review programme was not a realistic option. To solve the problem, the Monday was chosen as the fixed review day. The 'rules of the game' were as follows:

- The designers delivered one or more documents for review to the test manager before 9.00am (if no documents were delivered, no review occurred on that Monday).
- All documents taken together could not exceed a total of 30 A4-sized pages.
- The test manager determined which tester had to review what.
- The review comments of the testers had to be returned to the relevant designer before 12.00pm.
- The review meeting was held from 2.00pm to 3.00pm.

- The aim was for the designer to modify the document the same day (depending on the severity of the comments, one day later was allowed).

4.11.3 Inspections

There are various forms of the inspection process. This subsection describes a general form. For a specific, and indeed most common form, please refer to [Gilb, 1993] and [Fagan, 1986].

4.11.3.1 Introduction

A formal work method is followed when executing an inspection, with products being read thoroughly by a group of experts. This can be any of the documents previously listed in this section. The inspectors look at deviations from pre-defined criteria in these products, which must be 100% complete. In addition to determining whether the solution is adequately processed, an inspection focuses primarily on achieving consensus on the quality of a product. Aspects for evaluation are e.g. compliance of the product with certain standards, specifications, regulations, guidelines, plans and/or procedures. Often the inspection criteria are collected and recorded in a checklist. The aim of the inspection is to help the author find as many deviations as possible in the available time.

4.11.3.2 Responsibilities

Three to six participants are involved in an inspection. Possible roles are:

- Moderator
  The moderator prepares and leads the inspection process. This means planning the process, making agreements, determining product and group size, and bearing responsibility for recording all defects detected during the inspection process.
- Author
  The author requests an inspection. During the inspection process, he explains anything that may be unclear in the product. The author must also ensure that he understands the defects found by the inspectors.
- Minutes secretary
  The minutes secretary records all defects, actions, decisions and recommendations of the inspectors during the inspection process.
- Inspector
  Prior to the inspection meeting, the inspector tries to find and document as many defects as possible. Often an inspector is asked by the moderator to do so from one specific perspective, e.g. project management, testing, development or quality (see text box "perspective-based reading"). An inspector can also be asked to check the entire product on the correct use of a certain standard.

With the exception of the author, all participants may fulfill one or more of the above roles. All participants (except the author) at least fulfill the role of inspector. None of the participants may be the superior of one of the other participants because there is a risk that participants will restrain themselves when reporting their findings.

In more detail

Perspective-based reading
Participants in an evaluation activity often assess a product with the same objective and from the same perspective. Often, a systematic approach during preparation is missing. The risk is that the various participants find the same type of defects. Using a good reading technique can result in improvement. One technique commonly used is the perspective-based reading (PBR) technique. Properties of PBR are:

- Participants evaluate a product from one specific perspective (e.g. as developer, tester, user, project manager).
- Approach based on the *what* and *how* questions. It is laid down in a procedure *what* the product parts are that must be evaluated from one specific perspective and *how* they must be evaluated. Often a procedure (scenario) is created for each perspective.

PBR is one of the 'scenario-based reading' (SBR) techniques. Defect reading, scope reading, use-based reading and horizontal/vertical reading are other SBR techniques that participants in an evaluation activity can use.
Please refer to e.g. [Laitenberger, 1995] and [Basili, 1997] for more information.

4.11.3.3 Entry criteria

The inspection process can be started when:
- The aim of the inspection and the inspection procedure are clear.
- The product is 100% complete (but not yet definitive), and spelling errors have been eliminated, it complies with the agreed standards, accompanying documentation is available, references are correct, etc.
- The checklists and inspection forms to record defects to be used during the inspection are available.
- (Possibly) a list with known defects is available.

4.11.3.4 Procedures

The inspection process can be split up into the phases planning, kick-off, preparation and execution:

- *Planning*

  When the product for inspection complies with the entry criteria, the moderator organizes an inspection meeting. This means, among other things: determining date and location of the meeting, creating a team, allocating roles, distributing the products for inspection (delivered by the author) to the participants, and reaching agreements on the period within which the author must receive the defects found.

- *Kick-off*

  The kick-off meeting is held before the actual inspection. The meeting is optional and is organized by the moderator for the following reasons:
  - If participants are invited that have not yet participated in an inspection before, the moderator provides a summary introduction to the technique and the method used.
  - The author of the product for inspection describes the product.
  - If there are improvements or changes in the work method to be used during the inspection, they are explained.

  If a kick-off is held, the documents are handed out and the roles can be explained during this meeting.

- *Preparation*

  Good preparation is necessary to ensure the most efficient and effective possible inspection meeting. During the preparation, the inspectors look for defects in the products for inspection and record them on the inspection form. The moderator collects the forms, classifies the defects, and makes the result available to the author in a timely manner so that the latter can prepare.

- *Execution*

  The aim of the inspection meeting is not just recording the defects detected by the participants in the preparation phase. Detecting new defects during the meeting and the implicit exchange of knowledge are other important objectives. During the meeting, the moderator ensures that the defects are inventoried page by page in an efficient manner. The minutes secretary records the defects on a defect list. Cosmetic defects are not generally registered, but handed over to the author at the end of the meeting.

  At the end of the meeting, the moderator goes through the defect list prepared by the minutes secretary with all participants to ensure that it is complete and the defects are recorded correctly. Inaccuracies are corrected immediately, but due to efficiency the idea is not to open a discussion on defects and (possible) solutions.

  Finally, it is determined whether the product is accepted as is (possibly with some small changes), or if the product is accepted after changes and a check by one of the inspectors, or if the product must be submitted to re-inspection after changes.

  Based on the defect list and agreements made during the meeting, the author adapts the product.

4.11.3.5 Exit criteria

The inspection process is considered complete when:
- The changes (rework) are complete (check by moderator).
- The product has been given a new version number.
- All changes made are documented in the new version of the inspected product (change history).
- Any change proposals with respect to other products, that have emerged from the inspection process, have been submitted.
- The inspection form has been completed and handed over to the quality management employee responsible, among other things for statistical purposes.

4.11.4 Reviews

There are various review types, such as: technical review (e.g. selecting solution direction/alternative), management review (e.g. determining project status), peer review (review by colleagues), and expert review (review by experts). This subsection describes the review process in general.

4.11.4.1 Introduction

A review follows a formal method, where a product (60-80% complete) is submitted to a number of reviewers with the question to assess it from a certain perspective (depending on the review type). The author collects the comments and on this basis adjusts the product.

A review focuses primarily on finding courses for a solution on the basis of the knowledge and competencies of the reviewers, and on finding and correcting defects. A review of a product often occurs earlier on in the product's lifecycle than a product inspection.

4.11.4.2 Responsibilities

The minimum number of participants in a review is three. This may be less for a peer review, e.g. when reviewing the code, which is usually done by one reviewer. Possible roles are:
- Moderator

The moderator prepares and leads the review process. This means planning the review process, inviting the reviewers, and possibly allocating specific tasks to the reviewers.
- Author
  The author requests a review and delivers the product for reviewing.
- Minutes secretary
  The minutes secretary records all defects, actions, decisions and recommendations of the reviewers during the review process.
- Reviewer
  Prior to the review meeting, the inspector tries to find and document as many defects as possible.

All participants may fulfill one or more of the above roles. All participants (except the author) at least fulfill the role of reviewer.

4.11.4.3 Entry criteria

The review process can be started when:
- The aim of the review and the review procedure are clear.
- The product for review is available. A comprehensive "entry check" is not necessary because the product is only 60%-80% complete.
- The checklists and inspection forms to record defects to be used during the review are available.
- (Possibly) a list with known defects is available.

4.11.4.4 Procedures

The review process can be split up into the phases planning, preparation and execution:

- *Planning*

  When the product for review complies with the entry criteria, the moderator organizes an review meeting. This means, among other things: determining date and location of the meeting, inviting reviewers, distributing the products for review to the participants, and reaching agreements on the period within which the author must receive the defects found.

- *Preparation*

  Good preparation is necessary to ensure the most efficient and effective review meeting. During the preparation, the reviewers look for defects in the products for review and record them on the review form. The moderator collects the forms, preferably classifies the defects, and makes the result available to the author in a timely manner so that the latter can prepare.

- *Execution*

  At the beginning of the meeting, the agenda is created or adjusted under the leadership of the moderator. The most important defects are placed at the top of the agenda. The objective is to reserve adequate time in the agenda to discuss these defects. Since the product is not yet complete, little to no attention is devoted to less important defects (contrary to the inspection process). The minutes secretary records the defects on a defect list. Often an action list is also compiled.

  Finally, the moderator may recommend an additional review, determined among other things by the severity and number of defects. Based on the defect/action list made during the meeting, the author adapts the product.

4.11.4.5 Exit criteria

The review process is considered complete when:
- All actions on the action list are "closed" and all changes based on important defects are incorporated into the product (check by moderator).
- The product is approved for use in a subsequent phase/activity.

4.11.5 Walkthroughs

A walkthrough is a method by which the author explains the contents of a product during a meeting. Several different objectives are possible:
- Bringing all participants to the same starting point, e.g. in preparation for a review or inspection process.
- Transfer of information, e.g. to developers and testers to help them in their programming and test design work, respectively.
- Asking the participants for additional information.
- Letting the participants choose from the alternatives proposed by the author.

4.11.5.1 Introduction

A walkthrough can be organized for any of the mentioned documents when they are 50-100% complete.

4.11.5.2 Responsibilities

The number of participants in a walkthrough is unlimited if the author wishes to explain his product to certain groups, e.g. by means of a presentation. For an interactive walkthrough, we recommend a group size of two to seven persons. Possible roles are:
- Moderator
  The moderator prepares the walkthrough. This means planning the walkthrough, inviting the participants, distributing both the product and a document explaining the purpose of the walkthrough.
- Author
  The author requests a walkthrough and explains the product during the walkthrough.
- Minutes secretary
  The minutes secretary records all decisions and identified actions during the walkthrough. He also records findings (such as conflicts, questions and omissions) and recommendations from the participants.
- Participant
  The participant's role depends on the purpose of the walkthrough. It can vary from listener to actively proposing certain solutions.

All participants may fulfill one or more of these roles. The author can act as the moderator. Both the moderator and the author may fulfill the role of minutes secretary.

4.11.5.3 Entry criteria

The walkthrough can be started when:
- The purpose of the walkthrough is clear.
- The subject (product) of the walkthrough is available.

4.11.5.4 Procedures

The walkthrough process can be split up into the phases planning, preparation and execution:

- *Planning*

  When the entry criteria are met, the moderator plans the walkthrough, invites participants, distributes the product and explains the purpose of the walkthrough to the participants.

- *Preparation*

  Depending on the purpose of the walkthrough, the participants may submit defects but usually this does not happen (for example because the purpose is knowledge transfer). If defects are submitted, the moderator collects them and makes them available to the author. The author determines how the product is presented, e.g. relating to any defects, sequentially, bottom up or top down.

- *Execution*

  At the beginning of the meeting, the moderator explains the purpose of the walkthrough and the procedure to be followed. The author then provides a detailed description of the product; the participants can ask questions, submit comments and criticism, etc during or after the description.

  Decisions, identified actions, any defects, etc are recorded by the minutes secretary. At the end of the walkthrough, the moderator goes through the recorded decisions, actions and other important information with all those present for verification purposes.

4.11.5.5 Exit criteria

The walkthrough is considered complete when:
- The product has been described during the walkthrough.
- Decisions, actions and recommendations have been recorded
- The purpose of the walkthrough has been achieved.

4.11.6 Evaluation technique selection matrix

As with testing, every organization or project organizes evaluation processes in their own way. This means that there is no single uniform description of the evaluation techniques, nor can we specify in which situation a specific technique is most suitable. However, the table on the next page (also to be found at www.tmap.net) may offer some assistance in selecting a technique:

Aspect	Inspection	Review	Walkthrough
Area of application	In addition to determining whether the solution is adequately processed, focuses primarily on achieving consensus on the quality of a product.	Focuses primarily on finding courses for a solution and on finding and correcting defects. Review types include technical, management, peer and expert review.	Focuses on choosing from alternative solutions, completing missing information, or knowledge transfer.
Products to be evaluated	For example: functional/technical design, requirements document, management plan, development plan, test plan, maintenance documentation, user/installation manual, software, release note, test design, test script, prototype, and screen print.		
Group size	Three to six participants.	At least three participants.	Two to seven participants in alternative version to unlimited for presentation version.
Preparation	Strict management of the aspects to be evaluated by the inspectors. Defects (based on checklists, standards, etc) described by inspectors to be delivered to the author before the meeting.	Reviewers largely determine themselves which aspects they want to evaluate. Defects of reviewers to be delivered to the author before the meeting.	From being informed of the product to delivering defects.
Product status and size	Product is 100% complete, not yet definitive and limited in size (10-20 pages).	Product is 60%-80% complete and has a variable size.	Product is 50%-100% complete and has a variable size.
Benefits	High quality, incidental and structural quality improvement.	Limited labor intensity, early involvement of reviewers.	High learning impact, low labor intensity.
Disadvantages	Labor-intensive (costly), relatively long lead time.	Subjective, possible disturbance of collegial relationships.	Risk of ad-hoc discussions because participants are often not prepared.

4.12 Quality measures during development

In this section, a number of measures are described that can be used in, or be of influence on, the development tests. Most of these measures have consequences for the way in which the unit test and/or the unit integration test is carried out, apart from the code review which is an addition to the development tests and has no direct influence on these. It depends on the situation whether any of these measures, and if so which ones, will be chosen. For that reason, they are not included in the development test activities discussed in section 4.8, but are briefly described below. The fact that they are optional emphatically does not mean that they only have limited advantages. On the contrary, appropriate application of the measures in the right context can deliver huge advantages.

The following measures are discussed in succession:

- Test-Driven Development (TDD)
  TDD is a development method that strongly influences the UT, because it presupposes automated tests and ensures that test code is present for all the (source) code.
- Pair Programming
  A development method in which two developers work on the same software and also specify and execute the unit test in mutual co-operation.
- Code review
  This evaluation of the code supplements the development tests.
- Continuous Integration
  An integrative approach that requires automated unit and unit integration tests, minimizing the chance of regression errors.
- Agreed upon quality of development tests
  This approach is closely connected with, and forms part of, the test strategy. The choices made exert a strong influence on the method of specifying and executing the UT and UIT.
- Application integrator approach
  An organizational solution for achieving a higher quality of the UT and UIT.

4.12.1 Test-Driven Development (TDD)

Test-Driven Development, or TDD, is one of the best practices of eXtreme Programming (XP), see figure 77. TDD has a lot of impact on the way in which development testing (also outside of XP, particularly with iterative and agile development) is organized these days. It is an iterative and incremental method of software development, in which no code is written before automated tests have been written for that code. The aim of TDD is to achieve fast feedback on the quality of the unit.

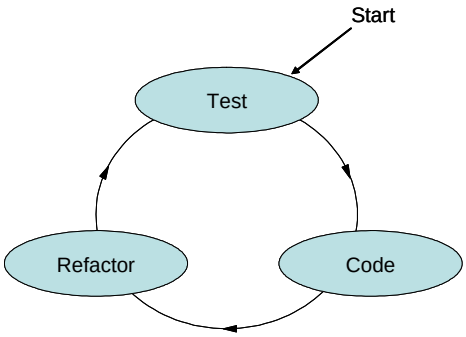


Figure 77. Test Driven Development.

Development is carried out in short cycles, in accordance with the above scheme:

Creating a test

- Write a test
  TDD always begins with the writing of test code that checks a particular property of the unit.
- Run the test (with fault)
  Execute the test and check that the result of the test is negative (no code has yet been written for that piece of functionality).

Encode and test

- Write the code
  Write the minimum code required to ensure that the test succeeds. The new code, written at this stage, will not be perfect. This is acceptable, since the subsequent steps will improve on it. It is important that the written code is only designed to allow the test to succeed; no code should be added for which no test has been designed.
- Execute the test and all the previously created tests
  If all the test cases now succeed, the developer knows that the code meets the requirements included in the test.

Refactoring

- Clean up and structure the code
  The code is cleaned up and structured without changing the semantics. The developer regularly executes all the test cases. As long as these are successful, the developer knows that his amendments are not damaging any existing functionality. If the result of a test case is negative, he has made a wrong change.

The application of TDD has a number of big advantages. It leads to:

- More testable software
  The code can usually be directly related to a test. In addition, the automated tests can be repeated as often as necessary.
- A collection of tests that grows in step with the software
  The testware is always up to date because it is linked one-to-one with the software.
- High level of test coverage
  Each piece of code is covered by a test case.
- More adjustable software
  Executing the tests frequently and automatically provides the developer with very fast feedback as to whether an adjustment has been implemented successfully or not.
- Higher quality of the code
  The higher test coverage and frequent repeats of the test ensure that the software contains fewer defects on transfer.
- Up-to-date documentation in the form of tests
  The tests make it clear what the result of the code should be, so that later maintenance is made considerably easier.

Tip

The theory of TDD sounds simple, but working in accordance with the TDD principles demands great discipline, as it is easy to backslide: writing functional code without having written a new test in advance. One way of preventing this is the combining of TDD and Pair Programming (discussed later); the developers then keep each other alert.

TDD has the following conditions:

- A different way of thinking: many developers assume that the extra effort in writing automated tests is disproportionate to the advantages it brings. It has been seen in

practice that the extra time involved in test-driven development is more than made up for by the gains in respect of debugging, changing the software and regression testing.
- A tool or test harness for creating automated tests. Test harnesses are available for most programming languages (such as JUnit for Java and NUnit for C#).
- A development environment that supports short-cycle test-code-refactoring.
- Management commitment to giving the developers enough time and opportunity to gain experience with TDD.

For more information on TDD, refer to [Beck, 2002].

In more detail

TDD strategy for testing GUI
TDD also has its limitations. One area where automated unit tests are difficult to implement is the Graphical User Interfaces (GUI). However, in order to make use of the advantages of TDD, the following strategy may be adopted:
- Divide the code as far as possible into components that can be built, tested and implemented separately.
- Keep the major part of the functionality (business logic) out of the GUI context. Let the GUI code be a thin layer on top of the stringently tested TDD code.

4.12.2 Pair Programming

Pair Programming is another best practice of eXtreme Programming (XP) that is also popular outside of XP. With Pair Programming, two developers work together on the same algorithm, design or piece of code, side by side in the same workplace.
There is a clear division of roles. The first developer is the one who operates the keyboard and actually writes the code. The second developer checks (evaluates!) and thinks ahead. While the code is being written, he is thinking about subsequent steps. Defects are quickly observed and removed. The two developers regularly swap roles.

The significant advantages of Pair Programming are:
- Many errors are caught during the typing, rather than during the tests or in use
- The number of defects in the end product is lower
- The (technical) designs are better and the number of code lines is lower
- The team solves problems faster
- The team members learn considerably more about the system and about software development
- The project ends with more team members understanding all parts of the system
- The team members learn to collaborate and speak to each other more often, resulting in improved information flows and team dynamic
- The team members enjoy their work more.

In recent decades, this method has been cited at various times as a better way of developing software. Research has demonstrated that by deploying a second developer in Pair Programming, the costs do not rise by 100%, as may be expected, but only by around 15% [Cockburn, 2000]. The investment is recouped in the later phases as a result of the shorter and cheaper testing, QA and management.

4.12.3 Code review

Another measure for increasing the quality of the developed products is an evaluation activity: the *code review*.

Definition

The code review is a method of improving the quality of written code by evaluating the work against the specifications and/or guidelines and subjecting it to peer review.

The code review can be carried out as an evaluation activity within development testing. Its aim is to ensure that the quality of the code meets the set functional and non-functional requirements.
In the code review, the following points can be checked, independently of the set requirements; see also figure 78:

1. Has the product been realized in accordance with the assignment? For example, are the requirements laid down in the technical design realized correctly, completely and demonstrably?
2. Does the product meet the following criteria: internally consistent, meeting standards and norms and representing the best possible solution? 'Best possible solution' means the 'best solution' that could be found within the given preconditions, such as time and finance.
3. Does the product contribute to the project and architecture aims? Is the product consistent with other, related products (consistency across the board)?
4. Is the product suitable for use in the next phase of the development (integration)?

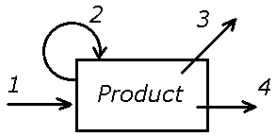


Figure 78. The 4 types of review goals and questions.

For the review, various techniques can be employed; see also section 4.12 "Evaluation techniques".

Tip

The code review is also applicable to development environments and development tools that are mainly used for configuration (rather than coding, e.g. with package implementations). In that case, the parameter settings are the subject of the review.

When carrying out the code review, allowance should be made for any overlap with the use of code-analysis tools; see section 4.8. These tools are increasingly being included in the automated integration process, in which they perform all kinds of analyses and checks automatically. The code review therefore does not have to focus on these. It is advisable to include the output of the tools in the code review report.

4.12.4 Continuous Integration

Continuous Integration has a particular influence on the organization of the unit integration tests. It is a way of working in which the developers regularly integrate their work, at least on a daily basis and increasing to several integrations per day. The integration itself, consisting of combining the units and compiling and linking into software, is automated. Each integration is verified by executing the automated tests in order to find integration defects as quickly as possible. The method minimizes the chances of regression faults. Continuous Integration ideally lends itself to combining with Test-Driven Development, requiring a development environment that supports automated integration and testing.

4.12.5 Agreed upon quality of development tests

An important reason for development testing is the meeting of the obvious expectation of the recipient party (client, project, system test, etc.) i.e. that the developed software 'simply' works. If many defects occur in the delivered software, this will cost (a lot of) time and money to solve. The developers get the blame for this and are accused of operating unprofessionally.

But what is obvious quality to the client(s)? It is wise to make these seldom-expressed expectations explicit. They can be roughly divided into obvious expectations in respect of skill, and obvious expectations in respect of product quality.

In more detail

In order to simplify the inventory, a summary of possible expectations is shown below.

Obvious quality of the product:

- Good is good enough
  The delivered product is not required to be perfect, but should be good enough to be transferred to the next phase (of testing). What often happens is that developers test the first units (too) thoroughly. They then come under pressure of time with later units and then do not test thoroughly enough.
- Once good, always good
  Changes to the product should not lead to lower quality of the total product, therefore regression faults are not tolerated.
- Processing of the most normal cases works flawlessly.
- Basic user-friendliness (e.g. standard validations, technical consistency, uniformity).

Obvious skill:

- Basic knowledge of working in projects
- Knowledge of the delivered quality
- Obligation to obtain confirmation of the assumptions (interpretations of the specifications)
- Obligation to signal "known errors" and/or "delay reports"
- Awareness of own inexperience and/or incapacity
- Optimum deployment of the available tools/facilities
- Enlisting support when in doubt.

Obvious product quality is important, but particularly difficult to establish. The product quality to be delivered is usually determined by the project, for which the developers work. This project, after all, has the purpose of delivering a product that works satisfactorily within a certain time and budget.

However, there is a footnote to this. What does the developer do if a project sets *no* specific requirements on the quality of the delivered software, neither in the master test plan, nor as suggested by the need for haste? Or even requests a "panic" delivery of the barely tested software? Are no development tests then carried out? At first sight, it seems acceptable to then leave out the development tests. If the project issues the order, delivery may take place without testing … however, experience teaches that it is extremely unwise to carry out no, or inadequate, development tests. Although the project will make the milestone at that point, a time will come during the system test or acceptance test, or even worse - in production, when the defects will stream in nevertheless. In the end, this reflects badly on the developers.

For that reason, the development department, for which the developers work, also bears a responsibility here. It can instruct that, irrespective of the project pressures, the developers should always deliver a consciously selected basic quality at minimum.
If the basic quality is clearly established, attention need only be paid in the formulation of the development testing task within a project to those parts of the system in which a higher level of certainly is required.

To this end, the developer (development department) should have the test intensity, clarity, provision of proof and compliance monitoring of the development tests established. An important decision to be made is the required degree of proof of the testing. How much certainty is required that the tests have actually been executed entirely in accordance with the agreed strategy? And how much time and money can be spared for providing this proof? Increasingly, external partners, too (e.g. supervisory bodies) are setting requirements on the proof to be supplied.

Example

Within an organization, basic quality is defined as follows:

Test intensity:
The basic test intensity is when all the statements in the realized software have been touched on in a development test at least once (statement coverage).

Clarity:
Enumeration of the situations under test with reference to the development basis (requirements, specifications, technical design), with indication of whether the test of the situation takes place in the code review, unit test or unit integration test.

Evidence:
In the list of situations under test, initialing to indicate what has been tested and by whom, without explaining how the testing was done.

Compliance monitoring:
Code reviews (random checks).

In addition to the basic quality, the development department may also opt for a model with several quality levels. This makes it easier for projects to make variations on the quality to be delivered.

Example

Over and above basic quality, an organization defines three other quality levels. The basic quality is given the label of *bronze* and the levels above it the labels *silver*, *gold* and *platinum*:

	Bronze	Silver	Gold	Platinum
Coverage thoroughness	Statement coverage	Condition/ decision coverage	Modified condition/ decision coverage	Multiple condition coverage
Clarity	Enumeration of the situations to be tested with reference to the test/development basis (requirements, specifications, technical design), indicating whether the test of the situation takes place in the code review, unit test or unit integration test	Test cases (logical), indicating whether the test takes place in the code review, unit test or unit integration test	Test cases (logical and physical), with indication of whether the test takes place in the code review, unit test or unit integration test	Test cases (logical and physical), with indication of whether the test takes place in the code review, unit test or unit integration test
Evidence	Initialed checklists	Test reports	Test reports + evidence	Test reports + evidence
Compliance monitoring	Code reviews and internal monitoring of test results (random checks)	Code reviews and internal monitoring of test results	Code reviews, internal monitoring of test results and random external audit checks	Code reviews, internal monitoring of test results and external audit

With the creation of several quality levels, a situation arises in which the client chooses the required quality for the various parts of the system, and the development department hangs a price tag on each quality level. In short, a first step towards negotiable selected quality. The selected quality is an agreement in respect of the formulation of the development tests in connection with the clarity, test intensity and proof of the executed tests.

Evidence

There are several possibilities for demonstrating that the required quality has actually been delivered (on time). Firstly, there is the provision of evidence option as established in the development testing strategy, with the exit and entry criteria. These should be met before the next test level starts. The project manager may also require additional evidence in order to monitor specific project risks. The decision regarding (additional) evidence involves weighing up experience, risks and associated costs.
Possible forms of evidence, which can often be combined, are:

- Marked/initialed test basis
  Initialing whatever has been tested in the development basis (requirements, functional design, use case descriptions and/or technical design), without indicating how the testing was done.
- Initialed checklist
  Deriving a checklist from the test basis (e.g. requirements, use case descriptions and/or technical design) and initialing what has been tested, without indicating how the testing was done.
- Test cases
  Test cases created using particular test design techniques with selected coverage thoroughness.
- Test cases + test reports
  As above, plus a report of which test cases have been executed, with what result.

- Test cases + test reports + evidence
  As above, plus evidence of the test execution in the form of screen and database dumps, overviews, etc.
- Test coverage tools (tools for measuring the degree of coverage obtained)
  The output of such tools shows what has been tested, e.g. what percentage of the code or of the interfaces between modules has been touched on. This can be a part of the build report.
- Automated tests
  The automated execution of tests in the new environment (e.g. system or acceptance test environment) very quickly demonstrates whether the delivered software is the same as the software that has been tested in the development tests and whether the installation has proceeded correctly.
- Demonstration
  Demonstrating that the (sub)functionality and/or the chosen architecture works according to the set requirements.
- Test-Driven Development compliance report
  By means of review reports, it can be demonstrated that the guidelines concerning the application of TDD have been met.

It should also be agreed how the reporting should be done. Perhaps a separate test report is to be delivered, or it may form a part of regular development reports.

4.12.6 Application integrator approach

If both the unit test and the unit integration test are employing a structured development test method, they should obviously be aligned, so that there are neither unnecessary overlaps nor gaps in the overall coverage of the development tests. A practical method is described below for simplifying this coordination, clearly stating the test responsibilities and offering an easy aid to the structuring of the development testing.

In this approach, an application integrator (AI) is given responsibility for the progress of the integration process and for the quality of the outgoing product. The AI consults with his client (the project manager or development team leader) concerning the quality to be delivered: under what circumstances may the system or subsystem be released to a subsequent phase (exit criteria). The AI also requests insight into the quality of the incoming units (entry criteria), in order to establish whether the quality of these products is sufficient to undergo his own integration process efficiently. A unit is only taken into the integration process if it meets the entry criteria. A (sub)system is issued if it (demonstrably) meets the exit criteria (see figure 79). It should be clear that the proper maintenance of exit and entry criteria has a great impact on (obtaining insight into) the quality of the individual units and the final system. Testing is very important for establishing these criteria, since parts of a criterion consist of, for example, the quality characteristic under test, the required degree of coverage, the use of particular test design techniques and the proof to be delivered. The entry and exit criteria are therefore used in determining the strategy of the unit test and the unit integration test. This method of operation also applies when the integration process consists of several steps, or in the case of maintenance.

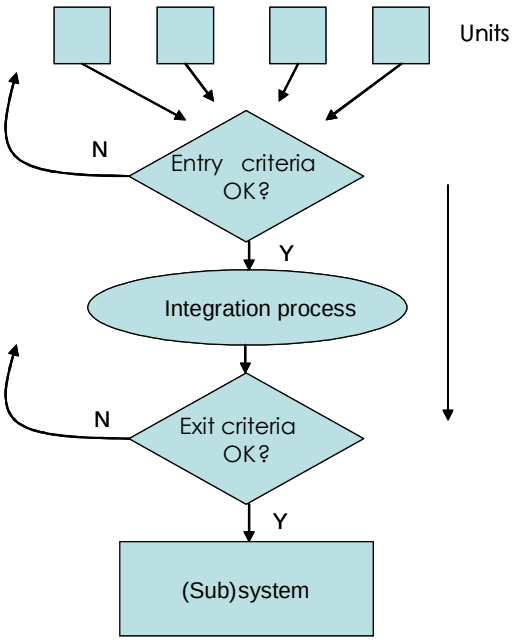


Figure 79. Entry and exit criteria.

To avoid conflicts of interest, ideally the AI does not simultaneously fill the role of designer or development project leader. This deliberately creates a tension between the AI, who is responsible for the quality, and the development project leader, who is judged in particular on aspects such as the delivered functionality, lead-time and budget spend.

Notable measures in the approach are:

- A conscious selection is made of the quality to be delivered and the tests to be executed before delivery to the following phase (so that insight is also obtained into the quality).
- The tests carried out by the developers become more transparent.
- Besides final responsibility on the part of the project or development team leader, the responsibility for the testing also lies with an individual inside the development team.

Implementations of this approach have shown that the later tests will deliver a lower number of serious defects. Another advantage of the approach is that earlier involvement of the system test and acceptance test is possible. Since there is improved insight into the quality of the individual parts of the system, a more informed consideration of risk can take place as regards earlier execution of certain tests. An example of this is that the acceptance test already evaluates the screens for user-friendliness and usability, while the unit integration test is still underway. Such tests are only useful when there is a reasonable degree of faith in the quality of these screens and the handling of them.

4.13 Introduction master test plan

Section "The role of testing" already discussed the fact that testing software (information system, package implementation or embedded software) is usually organized with a number of test levels. Each test level has a specific aim, e.g. establishing the correct functioning of a component or a system's adequate quality for production. When the test manager of each test level, in consultation with his direct clients, decides what will be tested, chances are that in the total picture of testing, certain matters will be tested twice unnecessarily, or that to the contrary specific issues are forgotten. The method should be

vice versa: based on the total overview a test manager, in consultation with the client and other stakeholders, makes a division as to which test level tests what and when (and with what intensity), fitting within the implemented development or maintenance approach. The aim is to detect the most important defects as early and economically as possible. This agreement is laid down in the so-called master test plan (MTP). This plan constitutes the basis for the detailed test plans for the separate test levels.
In addition to the content-based alignment, there are other reasons to try and gear activities to one another: ensuring uniformity in processes (e.g. the defect procedure and testware management), availability and management of the test environment and tools, and optimal division of resources (both people and means) across the test levels.

In more detail

Master test plan self-evident?
While creating a master test plan may seem self-evident, it is not that easy in actual practice. Unfortunately, the test managers of the individual test levels are often expected to achieve alignment. While agreement can usually be reached at the level of milestones, it becomes more difficult when involving the scope and thoroughness of the various test levels. A project manager is generally unable to devote adequate attention to this, while the test manager of a separate test level does not have the right authorizations. Fortunately, the importance of tuning and maintaining alignment of the test levels is understood more and more often in system development. For instance, in IBM's system development method, Rational Unified Process®[10], the master test plan is a formalized product.
Creating a master test plan and managing the total test process requires a specific role: the test manager or overall test coordinator for the overall test process.

In more detail

'Unnecessary double testing'
An important reason for a master test plan is preventing unnecessary double testing. The word 'unnecessary' is important in this context. As such, double tests are not a problem; often they cannot be avoided, and in some cases they are even mandatory. In a unit test, for instance, the same functionality will often be tested as in a system test, and various test cases will resemble each other. Part of the system test may also be repeated on purpose in the acceptance test, e.g. to check that everything works on the production-like infrastructure or within existing business processes. An example of 'unnecessary double' is when two test levels use a similar test design technique to derive and execute similar test cases.

This section discusses the management of the total test process based on the BDTM philosophy, and is therefore not limited to creating the master test plan. Coordination and adjustment is vital when executing the test plans. Delayed deliveries, disappointing quality, or a lack of time or resources are more of a rule than the exception in IT. The test manager of a test level must then adjust the planned approach. Coordinating and managing the total test process across the test levels must ensure that, despite individual adjustments in various test levels, a coherent overall approach for testing continues to be implemented. Adjustment in a test level may result in inefficiency in other test levels, e.g. because the test object has insufficient quality when completing the first test level. The test manager must notify others and propose or take measures in consultation with the client and project management. The so-called Deming wheel [Deming, 1992] can be distinguished in the

[10] Rational Unified Process is a registered trademark of IBM.

management and improvement mechanism used to this end: Plan something, Do it, Check the result, and Act if necessary. See figure 80.

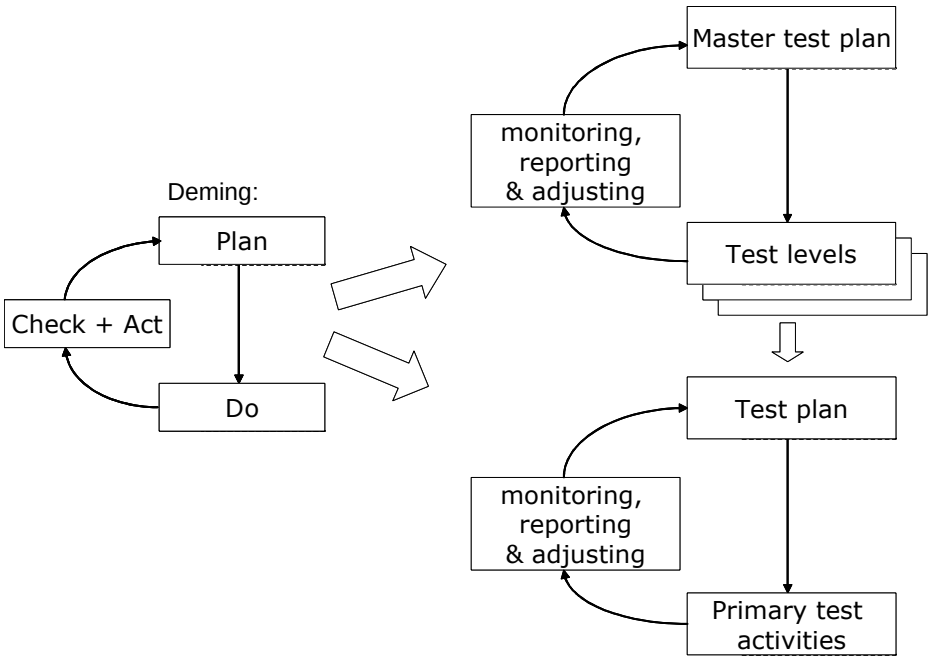


Figure 80. Deming wheel.

References

Exam literature

A **Workbook TMap® Suite**
Vianen, Sogeti Nederland BV, 2016

Additional literature

B Koomen, T., Aalst, L. van der, Broekman, B., Vroon, M.
TMap® Next, for result-driven testing
Vianen, Sogeti Nederland BV, 2014
ISBN 9789075414806 (ePub 9789075414486)
Note: This book has been published before under the same name by
Uitgeverij Tutein Nolthenius, 2007

C Boersma, A., Vooijs, E, Veltman, T.
Neil's Quest for Quality
A TMap® HD Story
Vianen, Sogeti Nederland BV, 2014 ISBN 9789075414837