Chapter 1
Effective testing

1.1 Testing information systems is becoming increasingly challenging

In everyday life we ‘test’ things quite often, for example, when buying new clothes, during wine-tastings and when we wish to purchase any particular type of consumer item. If you want to buy a car, you take a test-drive. You try out the turning circle, and you want to find out how quickly the car accelerates. You test the car to see if it lives up to your expectations, and should it fail to do so, you either go back to the salesman to ask if anything can be modified or you simply decide not to buy the car.

With information systems, you must also check whether your expectations are met. However, with information systems this process is much more complex and poses a considerable challenge to the test specialists. After all, information systems have a huge number of parameters, making the number of combinations so immense that they become practically untestable. A modern, object-orientated dialogue box for an end-user application will very soon possess two thousand characteristics, such as place, size, colour, activities to be undertaken, and underlying database activities. The complexity of information systems has increased recently due to the entrance of more and more new technologies, for example .NET technology. Such technologies offer a greater range of possibilities for the designers, but there is also a greater number of things that can go wrong. More links and interfaces are being added to the process, and more real-time systems are being employed. In the mainframe environment, batch processing is done for the most part by specialists, while with the client/server, responsibility for this task is shifting more and more towards the end user. With web applications, we do not even know who the users are. At the moment, the emphasis is on service-orientated programs, for example Software as a Service (SaaS), Test as a Service (TaaS) and Service Orientated Architecture (SOA). The complexity of this field is similarly on the increase. More and more individual services are offered that only work in conjunction with each other when one of these services sends out a request.

This makes testing both difficult and challenging, as in addition to having the necessary professional expertise you must also be both familiar with the environment and capable of preserving the interests of both the business and the IT department.
Society and the business world profits from good information systems, although we are becoming increasingly dependent on them. If an information system fails, it can have far-reaching consequences.

Should a chemist’s computer system turn out to be faulty, we would feel immensely unsafe, as this would entail much greater chances of somebody receiving the wrong medicines. The cause of the problem may lie in a failure in the automated prescription process (Van Moor, 2008).

Another famous, if older example is the failed launching of the Ariane 5 rocket. The rocket exploded mere seconds after launch, resulting in the wastage of billions of euros. The investigation showed that in the Ariane 5, software from the Ariane 3 was used without checking whether or not adjustments had to be made. The reason for this was that the deadline for the launch date had to be met.

When an information system works immediately, it often seems to happen more by accident than design. If you do not test the system properly, you can give no guarantees regarding the accuracy, security, reliability and user-friendliness of the information system. The company must therefore take testing of information systems very seriously, and approach it as a project. The time assigned for the testing must be sufficient to allow a thorough risk analysis. Based on this, you can make the right decisions regarding what you are going to test, you can prepare the tests, and you will still have enough time left to execute them. You must also be able to assign a sufficient number of people and resources. All of this costs money, although this investment will pay for itself very quickly. Just imagine what it would cost if a software defect comes to light after all customers or branches have already been supplied with the new software. The helpdesk would be swamped and unavailable for days, the programs would have to be recalled and adjusted, and these are merely the problems which can be expressed in hard currency. The damage to the company’s image may well be even more serious.

In the following section, we examine the ways in which testing can be approached. You can test in a number of ways: dynamic or static, on a running information system or on the basis of documents. These two methods are combined in the V model, the model which explains how development and testing can go hand in hand based on the different test levels. Subsequently, we expound black-box and white-box tests, which test external and internal aspects of the system. Finally, we discuss the various test levels and test types and the specific environments in which the tests are successively carried out. Once we have drawn up this overview, we will continue to Chapter 2 and begin to discuss TestFrame, the method that this book is all about.

1.2 Application of various methods of execution

You can test an information system in two ways, dynamically or statically. If you are going to perform a static test, this means that you are going to carry out the test without using the information system in question. Static testing encompasses all techniques concerning inspection and review of all documentation related to
1.2 Application of various methods of execution

the software in development. This also applies to the source code, the code of the program written by the programmers. Tools are often used for static testing of the source code. These tools supply information about different types of coverage, for example statement coverage and branch coverage. You therefore test the documents that were delivered during system development, as this documentation serves as the baseline for the test, the test basis.

Dynamic testing involves testing of the system while it is in operation. In this book, we will show that you can do this as soon as the first subsystems have been delivered. The preparation can even begin before the construction phase begins.

An example of static testing is the review of a requirements document, while an example of dynamic testing is running a computer program and checking that it accurately fulfils its intended function. We will now explore the difference between static and dynamic testing in greater depth.

1.2.1 Static testing

All phases in the system development process produce one or more documents, for example, requirements, system specifications and technical specifications. All of these ‘products’ must be tested for completeness and accuracy, by reading the specifications and assessing the content. This is called static testing.

Static testing is an efficient and effective method of preventing defects. After all, you execute a test immediately upon delivery of a requirement or design document, allowing you to discover defects in these documents before subsequent activities are started (Gilb & Graham, 1993).

If defects in the requirements or in the design are not discovered, then a requirement may, for example, be wrongly interpreted during programming. Experience tells that these sorts of defects are only discovered at a very late stage, for example, the acceptance test. The costs incurred in solving these defects are much greater than those incurred when the defects are found and solved during the design phase. If the defect is discovered during the acceptance test, then not only the software must be adjusted, but also the test cases, documents and instruction manuals. If the defect is caught during the design phase, only the design has to be altered. Furthermore, the larger the project, the more explosive the increase in costs in the event that a defect is found at an advanced stage (see fig. 1.1).

Therefore, the sooner the defect is discovered, the cheaper it is to fix. This example will illustrate this factor:

<table>
<thead>
<tr>
<th>Suppose that a user has defined the following requirements.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object: Car</td>
</tr>
<tr>
<td>Colour: Red</td>
</tr>
<tr>
<td>Seats: 2</td>
</tr>
<tr>
<td>Engine power output: 300 hp</td>
</tr>
</tbody>
</table>
Based on these requirements, at least two types of car could be produced, such as a red sports coupé and a red goods transit vehicle. This means that at least one requirement is lacking, and maybe more. If the requirement ‘Loading capacity’ was included with a value of above 3,500 kg, it would be clear that the sports car is not the valid option. However, a number of questions still remain, such as the question of whether a pick-up truck, a van or a heavy-goods vehicle is required.

Fig. 1.1  Explosive increase of costs to fix defects

There are many diverse forms of static testing. The difference between these forms is defined by the degree of formality and the phase of the project in which the static test takes place. The following forms of static test are involved (Gilb & Graham, 1993):

- **Informal review:** The main characteristic of an informal review (also known as a *collegial review*) is that it is not based on formal (documented) procedure. For this reason, the quality of an informal review can not be guaranteed. However, it is a relatively cheap way of quickly acquiring feedback.
- **Walk-through:** In a walk-through, the author gives a step-by-step presentation in order to collect information and to come to a common understanding of the content of the document. This allows him to receive feedback regarding the content
of the document, and gives indications for locating possible defects at an early stage in the project. Furthermore, this also gives his audience an opportunity to acquaint themselves with the new functionality.

- **Technical review:** During a technical review which may be led by an independent moderator, the process is examined by programmers, technical designers and architects after the necessary preparation has been carried out. The use of checklists may be helpful in this phase. The most important objectives of a technical review are detecting any possible defects, solving technical problems and determining the degree to which all requirements have been fulfilled.

- **Inspection:** An inspection is a formal procedure, led by a moderator, in which documents are visually inspected in order to detect defects. This process has clearly defined objectives, is well prepared and all anomalies are formally registered and processed. Based on the quality, further inspections may be planned.

### 1.2.2 Dynamic testing

In a dynamic test, the software of a particular component or system is put into operation. The test cases that are formulated for the various test levels during the analysis phase form the basis for this test. The objective is to determine whether or not the information meets the requirements and expectations.

For the execution of the tests, particular tools may often be necessary. In systems with a Graphical User Interface (GUI), values and actions are input on-screen, while in systems that do not have a user interface, tools must be employed, such as self-made utilities for data input, or standard tools for an SQL database.

In all dynamic tests, test cases are input into the information system and the results are recorded in a report.

For various dynamic tests, tools can be used for the purposes of automation, for instance, record and playback tools for executing functional tests.

### 1.2.3 Static and dynamic testing in the V model

In fig. 1.2, the V model is displayed. On the left-hand side of the V, in descending order, the successive phases are displayed that are generally involved in the development process of a system. On the right-hand side, the dynamic tests which correspond to the development activities on the left-hand side are shown.

Test levels consist of a combination of the static and dynamic tests that are found the same level. Example: early on in the process, the acceptance test begins with static testing of the users’ needs, requirements and business processes. If these are correct, the test analysts can begin to prepare the acceptance test. The actual dynamic execution of the acceptance test will only take place once the preceding test levels have been executed.
Below is a description of all links between system development and testing in descending order on the V model.

**Links between user preferences, requirements, business processes and the acceptance test**

To get a clear picture of exactly what the new information system must do, an information analyst or business consultant must map out the needs of the users and convert these, in conjunction with the business processes, into the business requirements. The test analysts will test these requirements statically for factors such as whether they are consistent, unambiguously formulated, testable etc. During dynamic testing, attention is focused on whether or not the information system indeed functions in accordance with what is stated in the documentation.

The acceptance test is executed in order to check whether the information system satisfies the users’ needs, requirements and business processes. Based on these test results, the users can accept or decline the information system. Distinction is often made between two sorts of acceptance tests: the User Acceptance Test (UAT) and the Production Acceptance Test (PAT). Of course, more forms of acceptance test are possible, but these two are most frequently employed. In the UAT, the end users check whether or not the system complies with the acceptance criteria that they defined. In the PAT, the system is tested to find out whether the system can be correctly implemented and how the system will behave in the production environment.
Due to the fact that the information system is tested as to whether it complies with the initial requirements during the acceptation test, the end users (in the broadest sense of the term, which also includes, for example, the operators and testware management personnel) play an important role in this test level. They are the most qualified to assess whether or not these systems are usable in practice. They often make use of the expertise of test analysts in order to decide what will be tested and how that will be done. The test executors and test analysts will therefore work in close cooperation with the end users.

**Links between system specifications and the system test/system integration test.**

The system designers translate the requirements into *system specifications*. Within these specifications is a meticulous description of how the requirements of the user organisation must be implemented. For this purpose, the specifications contain all necessary calculations, the screen layouts, and an explanation of how all of the different functions work. As a result, the system specifications are also known as *functional specifications*. During the static test of these documents, the completeness and the consistency with higher-level documentation is assessed. Has everything described in the requirements been incorporated into the system specifications? Will the information be interpreted correctly? Are the described functions, formulas etc. testable and unambiguous?

During the dynamic part of the *system test*, all of the information system’s functions are tested against the prescribed requirements and system specifications. The test preparation and specification can begin as soon as the first system specifications are provided, and therefore run parallel to the system development.

During the *system integration test*, the information system is tested to see how well it interfaces with the surrounding systems. During this test, the emphasis is on its connection with other systems both inside the organisation and out, such as the company back office or the Internet.

In the system test and the system integration test, we want to achieve as large a degree of independence as possible between test specialists and developers. By now, the components have all been connected, and in this phase the entire system is tested for the first time. An independent test group is the most capable of assessing whether or not the system functions in accordance with the prescribed specifications. Possible differences regarding interpretations of specifications can also be detected in this way.

**Links between code & technical design and the component test/component integration test.**

Based on the system specifications, the *technical design* is drawn up by the technical designers and programmers, and eventually the software (code) is written. The technical specifications contain details regarding the way in which specific aspects of the programs must be implemented.
During the component test, the test specialists will check whether the completed information system’s individual components (program modules) have been built in accordance with the technical design. The component test is principally focused on the internal program structure of the component and the functional aspects that can be tested at the component level. The component test can be performed both statically and dynamically. A static component test may be characterised by, for example, a static code analysis of the source code. In a dynamic test, components are put into operation, and stubs and drivers are often used in order to imitate the preceding or subsequent activities.

During the component integration test, tests are carried out to check whether two combined components form a working compound element of the information system. The emphasis of this test is on the communication channels or interfaces. Relationships between fields that are distributed across various components can be tested in the component integration test.

During the component and component integration test, it is often the case that no independent test specialists are appointed. The programmers often assume the role of test specialist, as construction and testing are very closely linked during this phase. If a programmer constructs a component, he will also continually check that this component functions correctly. If this proves not to be the case, he will immediately correct the component. When multiple components are produced, they are tested in their mutually connected function. To keep the testing process as independent as possible, the ‘four-eye’ principle is applied. This signifies that someone other than the programmer tests the components and their integration.

The testing roles and associated tasks, responsibilities, and powers that may come into effect during a testing project are described in appendix A.

1.3 Black-box and white-box testing

Besides the distinction between dynamic and static tests, which concerns the actual moment of testing, a distinction is also made between testing the system as a whole to assess its functionality, and testing it as a combination of different components which all contribute to this functionality. Initially, you assess the exterior of the information system. If you input something, does the output match your expectations? Secondly, you assess how well the system processes the data within the information system. The first method is referred to as black-box testing, and the second is referred to as white-box testing.

Black-box testing

Black-box tests do not concern themselves with the internal processes of an information system. You view the information system as a whole and test on an external basis. If you input something into the information system, the result must be correct and perform its predetermined function. You check the system based on interfaces, tables and screen displays, paying no attention to the program code or the modules that make up the information system. Black-box tests are chiefly applied during
functional testing, i.e. the system test and acceptance test. For this method, functional knowledge is necessary rather than technical knowledge.

For comparison: if you test drive a car, which has just had its brakes repaired, then you check that the brakes work by stepping on the brake pedal when driving and assessing whether the car stops within a safe distance. You are hardly likely to check the brakes by unscrewing the wheels and inspecting the brake blocks.

**White-box testing**

White-box tests focus on the internal processing of the individual components. During this type of test you may check, for example, whether or not the modules are correctly programmed and that the correct intermediate results are written to disk. Furthermore, you check whether programming standards have been followed and that the code is not needlessly complicated, or maybe even unnecessary. You also check that the correct modules are called up using the correct parameters. Therefore, to perform a white-box test, you need knowledge of the internal structure of the software. White-box tests are chiefly used during component tests and component integration tests.

For comparison: the mechanic that fixed the car from the previous example wishes to check the different components of the brake system. He must check components such as the brake fluid and the cables. Once this is done, he uses a brake tester to measure the braking force and uses this measurement to assess whether the brakes are working properly. He uses his knowledge of the structure of the braking system in order to check the brakes, rather than getting into the driver’s seat and stepping on the pedal.

### 1.4 Test levels and test types

The development process is split into test levels. These test levels must collectively cover the main objectives as described in the test strategy. Each level covers a particular area of the testing process. Within a test level, a second subdivision can be made between test types. Test types cover specific test objectives, for example, reliability, conversion, safety, performance, regression, stress and user-friendliness. The test types cover both functional and non-functional requirements. The ISO 9126 standard, which defines quality standards of software, is of course frequently used in addition to the described requirements as a basis for determining the necessary test types.

Not all test types can be employed in all test levels. For example, you cannot test the stress of a system during the component test. For a stress test, you need to have access to the entire system and/or certain interfaces or links to other systems, and this is only possible in the system integration test or the acceptance test.

During planning of the test process, certain test types (depending on the scale of the project) are often set up as subprojects within the project as a whole. Test specialists require special skills and tools in order to conduct particular test types.
1.5 Test environment

For the various test levels and test types, specific test environments are necessary. The nature and composition of these specific environments depend on the goal of the test. For a component test, for example, a technically adequate environment is necessary, but the content of the data is less important. During a system test or acceptance test, however, the data is of importance, as the objective of these tests is also to test the functionality and usability of the application.

Four logical environments must be established for the testing of an information system, specifically Development, Test, Acceptance and Production (DTAP). DTAP is essential in guaranteeing the handover and the quality of modifications to the system. An important objective of the DTAP principle is guaranteeing the quality and stability of the production environment. All of the environments perform their particular roles in the information system’s development and maintenance. These separate environments are therefore under separate leadership. The various software components must be transferred from one environment to another according to fixed procedures and arrangements.

We will now give a brief description of the four test environments. Further information regarding these environments can be found in Chapter 8.

- **Development.** In this environment, the components (program modules) are programmed and the component test and component integration tests are carried out.
- **Test.** The test environment is set up to allow the system test to be performed. The databases are set up in such a way that the functional test can be carried out. For the system integration test, this test environment is linked with the environments of the systems between which interaction must take place.
- **Acceptance.** Ideally, the acceptance test environment should be a copy of the production environment. All systems that operate in the production environment are installed, and if necessary, active. The databases contain actual data that has been anonymously transferred from the production files. In this environment, the end users and testware maintenance personnel validate this information in order to establish whether the system can be put into use.
- **Production.** This is the environment in which all programs are eventually installed. No tests are performed in this environment.

1.6 Structure of this book

This concludes the general introduction to the testing of information systems, the different ways to approach testing, the test levels and types, and the test environments. In Chapter 2, you will find an introduction to TestFrame, the method that this book is all about. In this chapter, we will discuss the phases within TestFrame and the test products within these phases. Chapter 2 ends with the cluster card, the beginning of the actual testing. The rest of the book principally addresses test
1.6 Structure of this book

analysis; how to ensure that the correct tests have been prepared and executed. Chapter 3 therefore contains information regarding the clear demarcation of independent parts of the test: the test clusters. Within the test clusters you define test conditions (Chapter 4), you elaborate the test conditions into test cases (Chapter 5), and then you execute the tests (Chapter 6).

Chapter 7 contains only the most important information regarding test automation (navigation). The reason for this is that the subject is so specialised, we are going to write a separate book about it. We conclude the book with a chapter about management of test products (Chapter 8). This factor is indeed crucial, as test products must be reusable.
TestFrame
An Approach to Structured Testing
Schotanus, C.C.
2009, XVI, 184 p., Hardcover
ISBN: 978-3-642-00821-4