

1 Introduction

Like never before, everyday life has become dependent on software and software-based systems. Most of today's appliances, machines, and devices are completely or at least partly controlled by software. Administrative proceedings in state agencies and industry, too, rely to a large extent on highly complex IT systems. Examples are the management of insurance policies, inventory control systems, biometric characteristics in passports and ID cards, and the electronic health chip card.

High dependency on software

This strong dependency on software requires ever higher investments in quality assurance activities to enable IT systems to perform reliably. Software testing is developing toward a specialized, independent field of study and professional discipline within the computer sciences.

*Software testing –
a professional discipline in its
own right*

Within the discipline of software testing, “test management” is of particular importance. Test management comprises classical methods of project and risk management as well as knowledge of the appropriate use of well-defined test methods. With this stock-in-trade, the test manager¹ can select and purposefully implement appropriate measures to ensure that a defined basic product quality will be achieved. In doing so, the test manager adopts an engineering approach.

Test management

Whereas today's project management training is well established, and while there are a tremendous number of study courses, training programs, and specialist literature to choose from, there has, until recently, been hardly any attempt at defining or standardizing the contents of training programs for the “software test manager”. In view of the increasing responsibility assumed by test managers in the execution of their job, this has been an unsatisfactory situation.

Training for test managers

With the ISTQB Certified Tester – Advanced Level – Test Manager we have, for the first time, developed an internationally recognized training and qualification scheme that defines training contents and qualification

*ISTQB Certified Tester –
Advanced Level –
Test Manager*

1. Note: For ease of reading, we use the male pronoun throughout this book where general references are made to persons. By no means is there any intention of gender bias or discrimination against women.

modules for the tasks of the test manager. This book sets out to convey the associated teaching contents and may be read as a textbook in preparation for the exams.

Foundation Level

The “ISTQB Certified Tester” qualification scheme consists of three levels. The basics of software testing are described in the syllabus for the Foundation Level ([URL: ISTQB] -Syllabi), whereas the corresponding subject matter is explained in detail in *Software Testing Foundations* [Spillner 07].

Advanced Level

The Advanced Level curriculum ([URL: ISTQB] -Syllabi) defines advanced proficiency skills in software review and testing and shows possible opportunities for specialization:

- Exhaustive treatment of different black box and white box test methods in the Advanced Level, Technical Tester, and Functional Tester modules
- Extensive, in-depth presentation of test management methods and techniques in the Test Manager module²

Since the “Advanced Level” syllabus is very comprehensive, it will not be treated in its entirety in this book; instead, we shall concentrate exclusively on the “Advanced Level – Test Manager module”. The topic of “reviews”³, however, will be excluded.

Expert Level

The third level, the “Expert Level”, is in the process of being defined by expert groups and comprises topics such as the specific characteristics of object-oriented software testing, advanced knowledge in Testing & Test Control Notation (TTCN-3, [URL: TTCN-3]), advanced knowledge in test process improvement methodology, and various other areas of expertise associated with software testing.

*International Software Testing
Qualifications Board (ISTQB)*

The “ISTQB” [URL: ISTQB] provides for the homogeneity and comparability of the teaching and examination contents of all participating countries.

Today, the ISTQB has become an affiliation of more than 25 national initiatives and associations worldwide (see figure 1-1). More national boards will follow.

-
2. The new version of the ISTQB Advanced Level syllabus is currently under development and will presumably adopt and advance this module structure.
 3. Regarding reviews see, for example, [Gilb 96].

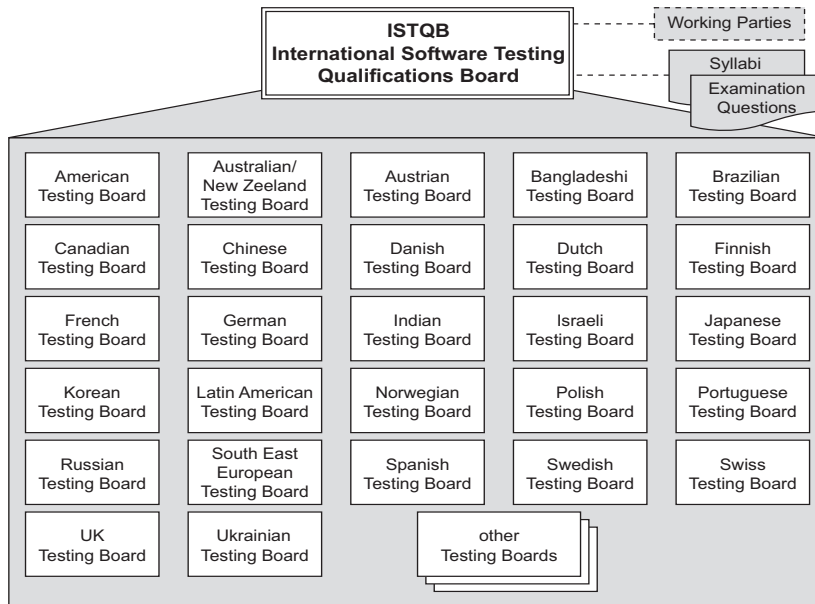


Figure 1-1
Structure of the ISTQB

As independent expert bodies, the national testing boards are responsible for the provision of training (accreditation of providers of proficiency testing schemes) and examinations (certification by an independent institution) in their respective countries and native languages and for ensuring compliance with ISTQB standards.

National Testing Boards

The three levels of the ISTQB qualification scheme build on one another. This book, *Software Testing Practice: Test Management*, presumes that the reader is familiar with the subject matters dealt with in the Foundation Level.

Knowledge of software testing foundations required

Readers new to software testing are advised to first acquire knowledge of the content of the Foundation Level, either by attending an accredited provider's seminar or by working through the book *Software Testing Foundations* [Spiller 07]. The present book contains only a brief recapitulation of the most important basic principles.

1.1 Software Testing Foundations – Condensed

This section provides a brief summary of the Foundation Level syllabus and of the book *Software Testing Foundations*.

*Measures to improve
software quality*

There is a multitude of approaches and proposals available on how to improve software quality through preventive (constructive) actions and the use of verifying (analytical) methods. The following measures are among the most important to improve software quality:

- Defined software development processes that contribute to a structured and traceable development of software systems
- A well-defined test process and controlled change and incident management as a requirement for the efficient and effective execution of test activities
- The application of metrics and quality data to objectively evaluate software products and development processes, to detect improvement potentials, and to verify the effectiveness of correction and improvement activities
- The use of formal methods that allow for the precise formulation of development documents and their verification or evaluation by tools
- Methods for the systematic identification and execution of test cases that allow efficient detection of defects and anomalies in the developed programs
- Methods for static testing, primarily reviews through which defects and anomalies are detected in design documents at an early stage

*Quality goals
and quality attributes*

Test managers must master or at least be familiar with these methods, techniques, and processes in order to be able to select and apply appropriate measures during the course of the project.

The suitability of quality assurance measures, however, also depends on the defined quality goals. The required quality level can thereby be defined based on different quality attributes. A catalogue of such quality attributes (e.g., functionality, reliability, and usability) is defined by the [ISO 9126] standard.

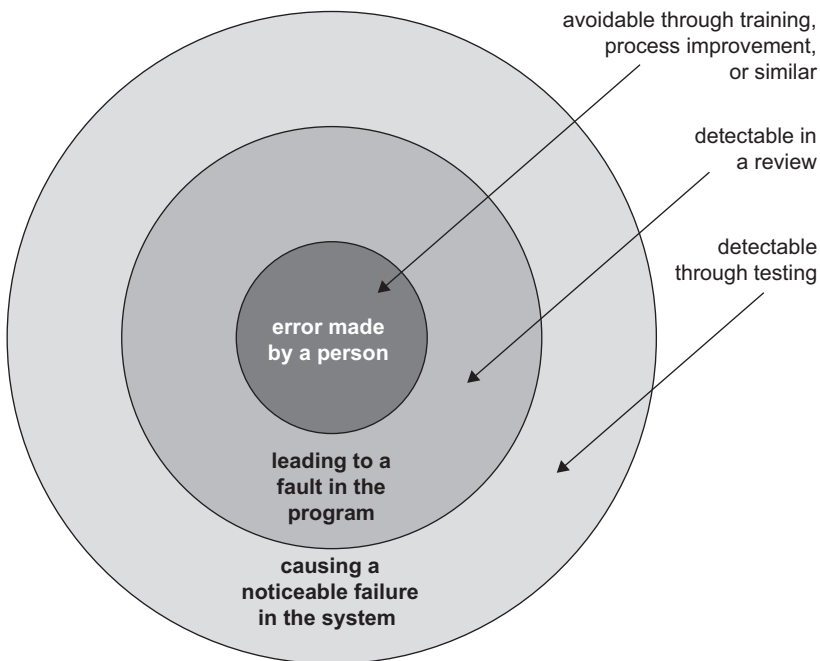
When do we speak of a defect or an error and what do we actually mean when we use these terms? A situation or result can only be classified as faulty, defective, or erroneous if we have previously defined what the expected, correct situation or result is supposed to look like. If the actual software behavior deviates from the expected behavior, we use words such as defect, fault, bug, and anomaly.

Test oracle

In order to establish expected values or expected behavior, a so-called test oracle is required that serves the tester as a source of information. Requirements documents, formal specifications, and the user guide are examples of such information sources.

The term “error” is actually rather imprecise. We do in fact need to distinguish between *error*, *fault*, and *failure* (including their synonyms). For example, a developer’s error while programming leads to a fault in the software that may (but not necessarily) result in a visible failure. In most cases, the impact of a fault or defect only shows itself in uncommon situations; for instance, the erroneous calculation of the leap year becomes effective only on February 29.

Figure 1–2 illustrates the relationship between error, fault, and failure and shows which countermeasures or methods may be used for their detection.



Error terminology

Figure 1–2
Relationship between the different terms used to denote errors

Similar to the term error, the word “test” has different meanings.

Testing frequently denotes the entire process of systematically checking a program to gain confidence in the correct implementation of the requirements⁴ and to detect failures. It is also a generic term for all the activities and (test) levels in the test process. Each individual execution of

Test terminology

4. Testing cannot prove that requirements are met 100% because it conducts only spot-check-like verifications.

	<p>a test object under specified conditions to verify the correctness of the expected results is also called testing.</p>
<i>Fundamental test process</i>	<p>Testing includes many individual activities. The following basic test process is defined in the Foundation Level syllabus, comprising the following activities:</p> <ul style="list-style-type: none"> ■ Test planning and control ■ Test analysis and test design ■ Test implementation and test execution ■ Test evaluation of the test exit criteria ■ Post testing activities
<i>Test levels</i>	<p>During testing, the product under test (the test object) can be considered at different levels of abstraction or on the basis of different documents and development products. The corresponding term is test level. We distinguish between the different levels of component test, integration test, system test, and acceptance test.</p> <p>Each test level has its own characteristic test objectives, test methods, and test tools.</p>
<i>Test types</i>	<p>Furthermore, we distinguish between different →test types: functional test, nonfunctional test, structural test, change-related test, and regression testing. [Spillner 07, section 3.7]</p> <p>During testing, we distinguish whether testing is performed by execution of the test object or whether it is done “only” on the associated program text or underlying specification or documentation.</p>
<i>Static and dynamic testing</i>	<p>In the first case, we have the so-called dynamic tests, represented by black box and white box test methods [Spillner 07, chapter 5], and in the second case, we talk about static tests, represented, among other things, by different types of reviews [Spillner 07, chapter 4].</p>
<i>Independence between test and development</i>	<p>Regardless of which method is used for testing, it is essential that in terms of organization, development/programming and testing are kept separate and that they are performed independently from each other. A developer testing his own code will be blind toward his own mistakes and not very keen on detecting them himself.</p>
<i>Test tools</i>	<p>There are many supporting tools in use for software testing. Depending on their intended use, we distinguish between different tool classes, such as, for instance, tools for test management and control, tools for test specification, and tools for static, dynamic and nonfunctional testing [Spillner 07, chapter 7].</p>

In our discussion of the Foundation Level syllabus, we reviewed the test management fundamentals. In addition to test planning, test control, and test reporting, test management includes topics such as change and configuration management as well as the economy of testing [Spillner 07, chapter 6]. This book will cover these test management tasks in more detail.

For illustration purposes, we shall continue the case study example introduced in the book *Software Testing Foundations*:

A car manufacturer develops a new electronic sales support system called *VirtualShowRoom (VSR)*. The final version of this software system will be installed at every car dealership worldwide. Customer interested in buying a new car will be able to configure their favorite model (model, type, color, extras, etc.) at the terminal with or without the guidance of a salesperson.

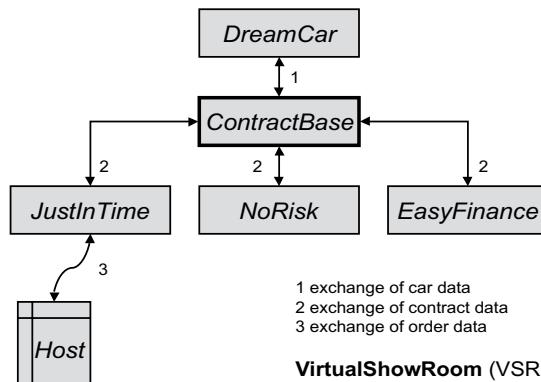
The system shows all possible models and combinations of extra equipment and instantly calculates the accurate price of the configured car.

This functionality will be implemented by a subsystem called *DreamCar*.

When customers make up their mind, they will be able to calculate the most suitable payment method (*EasyFinance*) as well as place an order online (*JustInTime*). Of course, they will have the opportunity to sign up for the appropriate insurance (*NoRisk*).

Personal information and contract data about the customer is managed by the *ContractBase* subsystem.

Figure 1-3 shows the general architecture of this software system.



Test management

Case study "VirtualShowRoom" (VSR)

Figure 1–3
Architecture of the VSR-System

Each subsystem will be designed and developed by separate developer teams.

Altogether, about 50 developers and additional employees from the respective user departments are involved in working on this project. External software companies are also involved.

In *Software Testing Foundations*, we described the different →test design techniques used for in-depth testing of the VSR system before putting it into operation.

VSR-2 development follows an iterative development process. Based on the current VSR-1 system, VSR-2 is supposed to be developed in four successive iterations. The planned schedule is one year, with approximately one increment per quarter. Each new increment is expected to provide the full functionality of the previous version; it may, however, be based on a better or more efficient implementation. In addition, each increment introduces a set of new functionalities.

The product manager expects two things from the test manager:

- First, the test team must ensure that the old functionality is correctly retained in each intermediary VSR-2 version.
- Second, the test team should fairly quickly be able to judge objectively if, and how well, a new feature has been implemented.

The tasks that the test manager has to perform regarding such issues will be discussed and illustrated in the following chapters.

1.2 Software Testing Practice: Test Management – Overview

Software testing practice – overview

The topics of the book and the contents of each chapter are briefly described here.

- Chapter 2 discusses the fundamental test process and the types of tools that can be used to support it. Both topics were covered in *Software Testing Foundations*.
- Chapter 3 explains how testing is related to the software life cycle. It discusses different life cycle models used in software development and evaluates the particular importance given to testing in each model.
- The significance given to testing in an organization is of great importance to the test manager. The organization's quality and test policy must be defined by management. Chapter 4 deals with these issues.
- Chapter 5 takes a closer look at test planning, one of the important, if not most important, tasks of the test manager.
- Planning must be adjusted during the project's life cycle. Control of the test process based on test progress reports is an essential task for the test manager in order to perform successful testing. This aspect is addressed in chapter 6.

- The development and test processes themselves can be evaluated and improved. Chapter 7 describes which techniques and processes are to be applied to accomplish this improvement.
- How do we deal with deviations and failures detected during testing? Chapter 8 provides some answers.
- Risk evaluation and risk-based tests are important instruments for the test manager to allocate limited test resources. They are used to control the test project with minimized risk. Chapter 9 contains advice on how to proceed.
- Without qualified and skilled staff – that is, without consideration of the human factor – the test manager will not be able to succeed. Chapter 10 describes what needs to be considered when selecting a test team.
- Test metrics help in defining test exit criteria and in finding evidence on the quality of the test object. Chapter 11 will provide some examples.
- In most cases, the test process can be performed more efficiently with adequate tool support. Chapter 12 describes how the test manager selects and introduces such tools.
- The last chapter, chapter 13, presents relevant standards.

The glossary contains all the terms that are newly mentioned in this book, the first occurrence of which will be preceded by an arrow “→”. All glossary terms used in *Software Testing Foundations* [Spillner 07] can also be found at [URL: ISTQB] -download.

2 Test Process and Test Tools

This chapter introduces the fundamental test process, its associated activities, and appropriate tool support¹.

2.1 Test Process Fundamentals

In order to perform structured tests, a general description of the task as found in most development models (see chapter 3) is not sufficient. Besides integrating testing into the development process it is also necessary to provide a detailed test procedure (see figure 2-1). The development task consists of the process phases test planning and control, analysis and design, implementation and execution, evaluation of the test exit criteria and reporting, as well as test completion activities. Although the presentation and description of the individual tasks suggest a sequential procedure they may of course overlap or be performed in parallel.

2.1.1 Test Planning and Control

Planning a comprehensive task such as testing ought to start as soon as possible in the initial stages of software development.

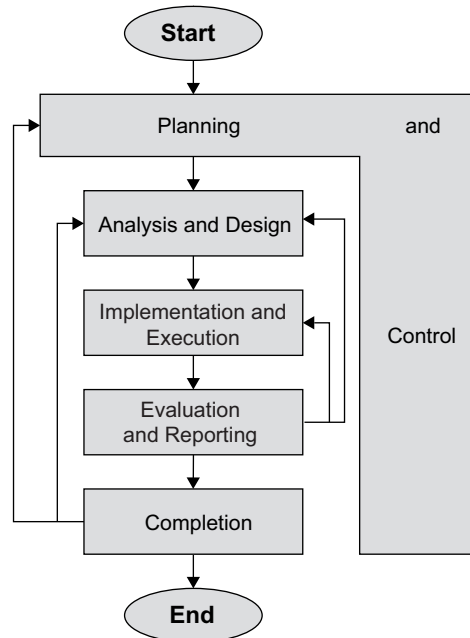
Resource planning

The role and purpose of testing must be defined as well as all the necessary resources, including staff for task execution, estimated time, facilities, and tools.

The associated specifications are to be documented in the test plan. An organizational structure including test management needs to be in place and ought to be adapted, if necessary.

1. A more detailed introduction to the fundamental test process is given in [Spillner 07, section 2.2 and chapter 7] Readers familiar with its content may skip this chapter.

Figure 2-1
Fundamental test process



Test management is responsible for the administration of the test process, the test infrastructure, and testware. Regular control is necessary to see if planning and project progress are in line. This may result in the need for updates and adjustments to plans to keep the test process under control. The basis for controlling the test process is either staff reporting or relevant data and its evaluation by appropriate tools.

Determining the test strategy

Since exhaustive testing is impossible, priorities must be set. Depending on the risks involved, different test techniques and test exit criteria must be specified when establishing a test strategy. Critical system components must be intensively tested. However, in the case of less critical components a less comprehensive test may suffice or testing may even be waived. The decision must be very well-founded to achieve the best possible allocation of the tests to the “important” parts of the software system.

Determining test exit criteria

→ Test intensity is determined by the test methods employed and by the intended degree of coverage when executing the test cases. The degree of coverage is one of several criteria for deciding when a test is completed.

Prioritizing tests

Software projects are often under pressure of time, a fact which must be anticipated during planning. Prioritizing tests causes the most critical

software components to be tested first in case not all planned tests can be performed due to time or resource constraints.

Without adequate tools the test process cannot be sufficiently performed. If tools are missing, their selection and procurement must be initiated early in the process.

Tool support

Moreover, parts of the test infrastructure themselves often need to be established, for instance the test environment, in which system components can be executed. They need to be put in place early so that they are available when coding of the test objects is completed.

2.1.2 TestAnalysis and Design

The test strategy developed during planning defines the test design techniques to be used. As a first step of test analysis, the test basis needs to be checked to see if all required documents are detailed and accurate enough to be able to derive the test techniques in agreement with the test strategy². The specification of the test object determines its expected behavior. The test designer uses it to derive the prerequisites and requirements of the test cases.

Verification of the test basis

Depending on the analysis results it may be necessary to rework the test basis so that it can serve as a starting point for the test design techniques defined in the test strategy. For example, if a specification is not accurate enough it may need to be improved. Sometimes it is the test strategy itself which may need to be changed, for instance, if it turns out that the selected test design techniques cannot be applied to the test basis.

During test design, test techniques are applied to identify the respective test cases, which are then documented in the test specification. Ultimately, the →test project or test schedule determines the timing of the test execution sequence and the assignment of the test cases to the individual testers.

When specifying test cases, logical test cases must be defined first. Once this is done, concrete, i.e., actual input and expected output, values may be defined.

Logical and concrete test cases

However, this is done during implementation, which is the next step of the fundamental test process.

2. This check can already be done while developing the test strategy and may influence it if the documentation is already available at that time.

<i>Black box and white box techniques</i>	Logical test cases can be identified based on the specification of the test objects (black box techniques) or based on program text (white box techniques). Thus, the specification of the test cases may take place at quite different times during the software development process (before or after or parallel to coding, depending on the test techniques selected in the test strategy). In this connection, many of the life cycle models described in the next chapter show only test execution phases (e.g., the general V-model). Test planning and specification activities can and should take place concurrently with earlier development activities, as explicitly pointed out in the W-model or in extreme programming.
<i>Test cases comprise more than just test data</i>	During test case specification the particular starting situation (pre-condition) must be described for each test case. Test constraints to be observed must be clearly defined. Prior to test execution it needs to be defined in the post-condition which results or which behavior is expected.
<i>Test oracle</i>	In order to determine the expected results a test oracle is queried which predicts the expected outcomes for every test case. In most cases the specification or the requirements are used as the test oracle to derive the expected results from individual test cases.
<i>Positive and negative test cases</i>	Test cases can be distinguished according to two criteria: <ul style="list-style-type: none"> ■ Test cases for testing specified results and reactions to be delivered by the test object (including treatment of specified exceptional and failure situations) ■ Test cases for testing the reaction of the test objects to invalid or unexpected inputs or other conditions for which “exception handling” has not been specified and which test the test object for robustness
<i>Setting up the infrastructure</i>	The required test infrastructure to run the test object with the specified test cases is to be established in parallel to the other activities so as to prevent delays in the execution of the test cases. At that point the test infrastructure should be set up, integrated, and also tested intensively.

2.1.3 Test Implementation and Execution

In this step of the test process, concrete test cases must be derived from the logical test cases, and executed. In order to run the tests, test infrastructure and test environment must both be implemented and in place. The individual test runs are to be performed and logged.

Timing and test case sequence The actual tests are to be run observing the priorities that we defined earlier. It is best to group individual test cases into test sequences or test

scenarios in order to allow for the tests to be run efficiently and to gain a clear structure of the test cases.

The required test harness must be installed in the test environment before the test cases can be executed.

At the lower test levels, component and integration testing, it makes sense to run automated rather than manual tests (e.g., using JUnit [URL: JUnit])

During test execution an initial check is done to see if the test object is, in principal, able to start up and run. This is followed by a check of the main functions (“smoke test” or acceptance test during entry check of the individual test levels).

Checking main function completeness

If failures occur already at this stage further testing makes little sense.

Test execution must be logged accurately and completely. Based on test protocols, test execution must be traceable and evidence must be provided that the planned test strategy has actually been implemented. The test protocol also contains details concerning which parts were tested when, by whom, to what extent, and with what result.

Tests without a test protocol are useless

With each failure recorded in the test log a decision needs to be made whether its origin is thought to lie inside or outside the test object. For instance, the test framework may have been defective or the test case may have been erroneously specified.

Evaluating the test protocols

If a failure exists it needs to be adequately documented and assigned to a incident class.

Based on the incident class the priority for defect removal is to be determined. Successful defect correction needs to be ascertained: has the defect been removed and are we sure that no further failures have occurred?

Correction successful?

The earlier made prioritization has the effect that the most important test cases are executed first and that serious failures can be detected and corrected early.

Most important tests come first!

The principle of equal distribution of limited test resources over all test objects of a project is of little use since such an approach leads to equally intensive testing of critical and non-critical program parts.

2.1.4 Test Evaluation and Test Report

It needs to be checked if the test exit criteria defined in the plan have been met. This check may lead to the conclusion that test activities may be considered completed but may also show that test cases were blocked and that

Test completion reached?

not all planned test cases could be executed. It may also mean that additional test cases are required to meet the criteria.

Closer analysis, however, may reveal that the necessary effort to meet all exit criteria is unreasonably high and that further test cases or test runs had best be eliminated. The associated risk needs to be evaluated and taken into account for the decision.

If further tests are necessary, the test process must be resumed and the step has to be identified from where test activities are to be resumed. If necessary, planning must be revised as additional resources are required.

Besides test coverage criteria, additional criteria may be used to determine the end of the test activities (see chapter 11).

Allow for several test cycles

Test cycles develop as a result of observed failures, their correction, and necessary retesting. Test management must take such correction and test cycles into account in their planning (see also the W-model in section 3.4). Otherwise, project delays are the rule. It is rather difficult to calculate the effort needed for the test cycles in advance. Comparative data from earlier, similar projects or from already completed test cycles may help.

*Exit criteria in practice:
time and cost*

In practice, time and cost often determine the end of testing and lead to the termination of test activities.

Even if during testing there is more budget spent than planned, testing as a whole does cause savings through the detection of failures and subsequent correction of software defects. Defects not detected here usually cause considerably higher cost when found during operation.

Test report

At the end of this activity of the test process, a summary report must be prepared for the decision makers (project manager, test manager, and customer, if necessary) (see also [IEEE 829]).

2.1.5 Completing the Test Activities

Learning from experience

Unfortunately, in practice, the closing phase of the test processes is mostly neglected. At this stage, the experiences gained during the test process should be analyzed and made available to other projects. In this connection, the presumed causes of differences between planning and implementation are of particular interest.

A critical evaluation of the activities performed in the test process, taking into account effort spent and the achieved results, will definitely reveal improvement potential. If these findings are documented and applied to subsequent projects in an understandable manner, continuous process

improvement has been achieved. Chapter 7 will take a closer look at the models for analysis, evaluation, and test process improvement.

A further finishing activity is the “conservation“ of the testware for future use. During the operational use of software systems, hitherto undetected failures will occur despite all previous testing, or customers will require changes. In both cases this will lead to revised versions of the program and require renewed testing. If testware (test cases, test protocols, test infrastructure, tools, etc.) from development is still available, test effort will be reduced during the maintenance or operational phases of the software.

Testware “conservation”

2.2 Test Tools

The following section provides an overview over different types of test tools³. Tool types are comprehensively described in *Software Testing Foundations* ([Spillner 07, chapter 7]). A closer look is taken at tools that support test management. It is particularly important for the test manager to learn how to select and use such tools (see chapter 12).

Short overview

There are many tools supporting or automating test activities, all of which are known as CAST tools (Computer Aided Software Testing), analogous to CASE tools (Computer Aided Software Engineering).

CAST tools

Depending on which activities or phases in the test process are supported, we may distinguish between different tool types.

As a rule, not all available test tools are applied in a project. However, the test manager should know available tool types in order to be able to decide if and when to use a tool efficiently in a project.

The following sections describe the various functions provided by the different tool types.

2.2.1 Tools for Management and Test Control

Planning and control are the first activities in the test process. The respective test management tools offer mechanisms for easy capturing, prioritizing, cataloguing, and administration of test cases. They allow test case status tracking, i.e., they document and evaluate if, when, how often, and with which result a test case has been executed. Moreover, they may be used to

Test management tool

3. [URL: Tool-List] provides a list of useful links regarding tool information.

Support for advanced management tasks

support resource and schedule planning for the tests. The test manager can plan the tests and remain informed at all times about the status of hundreds or thousands of test cases.

In addition to these core tasks, test management tools offer support for tasks and activities such as:

- Requirements-based testing: system requirements are linked with those tests that check the corresponding requirement. Different consistency checks can be performed, for instance to see if for each requirement at least one test case has been planned.
- Defect management: tool support is indispensable for the management of problem or incident reports. Capturing, administration, and statistical evaluation of incident reports should not be done manually, since this is simply too error-prone. Tools help the test manager to be kept informed about the actual project at all times.
- Preparing test reports and test documents: both test management and incident management tools provide extensive analysis and reporting features, including the possibility to generate complete test documentation (test plan, test specification, test report) from their data repository.

2.2.2 Tools for Test Data and Test Script Specification

Test data and test script generators

So-called test (data) generators can support the test designer in generating test data. There are several approaches, depending on the test basis used:

- Database-based test data generators create test data on the basis of database schemas or database content.
- Code-based test data generators analyze the source code to derive the test data. Target or expected values, however, cannot be derived.
- Interface-based test data generators derive test data through identification of interface parameter domains (for example, using equivalence class partitioning and boundary value analysis). Here, too, the problem exists that target values cannot be generated.
- Specification-based test data generators derive test data and associated target values from a formal specification.
- Model-based test generators derive test scripts from formal models, starting, for instance, from a UML sequence diagram specifying the call sequences of methods.

2.2.3 Tools for Static Testing

Static checks can be carried out on design documents, given the availability of a formal notation, and on (also only partially available) source code, i.e., even before executable program (parts) are available. Tools supporting the static test help to detect defects and discrepancies already in the early phases of the development process. Therefore a test manager should consider using these tools.

Static analysis

- Static analyzers provide measures of miscellaneous characteristics of the program code which can be used to identify complex and hence defect-prone or risky code sections. Violations of programming guidelines, broken or invalid links in website contents and many other anomalies or discrepancies can be analyzed statically.
- Model checkers analyze specifications if they are available in a formal notation or as a formal model. For example, they can find missing states, missing state transitions, and other inconsistencies in the state model to be checked.
- Furthermore, there are tools to support reviewing and which help in the planning, execution, and evaluation of review results.

2.2.4 Tools for Dynamic Testing

Test tools for automating test execution relieve the tester from carrying out unnecessary mechanical tasks. The tools supply the test object with test data, record the test object's reactions, perform a comparison with the expected reactions, and log the test execution.

Tool support for functional tests

- In the narrow sense, a debugger is not a test tool but is very useful for root cause analysis and for enforcing exception handling in the program code.
- Test drivers and test bed generators offer a mechanism to address test objects via their application programming interface (API), or via another interface not directly accessible to the end user, such as, for example, the Ethernet, serial interface, and so on.
- Simulators can be used to emulate an operating environment. They are particularly used for testing embedded software if the target system is not yet available, or if testing in the target system is very expensive or if it requires a disproportionately high effort.

- Test robots (capture and replay tools) record all input that is manually performed inputs (keyboard inputs, mouse clicks) during a test session and save them in a test script. The recorded test can be automatically repeated by replaying the test script as often as one likes.
- Comparators are used to identify differences between expected and actual results. They are typically included in other tools.
- Dynamic analyzers acquire additional information during program execution, for instance on allocation, usage, and release of memory (memory leaks, pointer allocation, pointer arithmetic problems, and so on).
- Coverage analyzers provide structural test coverage values measured at code level during test execution (see also chapter 11).

Tool support for non-functional tests

Besides tools that support functional testing there are also tools for testing non-functional features of the test objects:

- Load and performance test tools generate a synthetic load, e.g., database queries, user transactions, or network traffic, for the execution of volume, stress, and performance tests.
- Monitors are used to support tests and analysis in that they identify and evaluate necessary data. They are typically integrated in load and performance test tools.
- Tools for checking access and data security analyze possible security gaps in the test object.

2.2.5 Constraints to be Considered

Tool use and test process

Creative test activities can be supported by tools. The mechanical test execution can be automated, reducing the test effort or allowing the execution of more test cases without spending any additional effort. More test cases, however, do not necessarily mean better tests.

Without a well-established test process and adequate test methods, tools cannot achieve the desired cost reduction. The introduction and efficient use of tools requires a thorough evaluation of the test processes and accompanying process improvement activities (see also chapters 7 and 12).

On the other hand, the economic execution of the test processes can only be achieved with appropriate tool support; for instance, to be able to execute and evaluate many test cases within an adequate time frame.

The test manager must be aware of all these constraints and must act accordingly.

2.3 Summary

Testing must be divided into individual process steps. A fundamental test process is divided into the following steps:

- Test planning and control: Define required resources (staffing, schedule, tools), define the test strategy together with the selection of the test methods to be used, the respective coverage criteria, and prioritization of the tests. Also, determine the sequence of test execution in the test schedule. Intervene to control during the whole test processes if there are any deviations from the plan.
- Test analysis and design: Check the test basis for completeness and sufficient accuracy. Design logical test cases using the test methods of the test oracle. Begin creating the test infrastructure.
- Test implementation and execution: Draw up test cases and group them to test sequences or scenarios, and complete the test infrastructure. The first step in the execution is to check that the test object is executable and that calling up main functions does not cause any serious failures. All test runs are to be recorded and evaluated in detail.
- Test evaluation and report: Show that the test exit criteria have been satisfactorily fulfilled. If not, decide if further tests are to follow or if the test process may be finished nonetheless. Draw up a summary test report.
- Completion of the test activities: The main task of this last activity of the test process is to learn from experience and to provide the testware needed for maintenance.
- For each phase of the test process tools are available that help the test manager and the tester to qualitatively improve their test activities.
- The use of test tools is only of advantage if the test process is a controlled and defined process.