CHAPTER

2

Testing Processes

Contents

Everything we do, from cooking a meal to producing the most complicated software products, follows a process. A process is a series of activities performed to fulfill a purpose and produce a tangible output based on a given input.

The process view on software development is gaining more and more interest. Process models are defined to assist organizations in process improvement—that is, in making their work more structured and efficient.

Testing can also be regarded as a process.

Like all processes the test process can be viewed at different levels of detail. An activity in a process can be seen as a process in its own right and described as such. The generic test process consists of five activities or processes. Each of these is treated like a separate and complete process.

Test development is what is usually understood as the real test work. This is sometimes divided into two subprocesses, namely:

- Test analysis and design;
- Test implementation and execution.

The borderline between the two subprocesses is blurred and the activities are iterative across this borderline.

The two subprocesses are, however, discussed individually here.

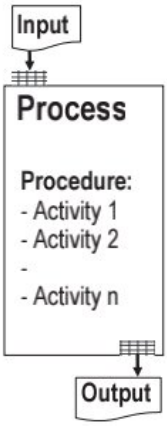
2.1 Processes in General

2.1.1 The Concept of a Process

A process is a series of activities performed to fulfill a specific purpose. Based on an input to the process and following the activities—also called the procedure—a tangible output is produced.

It is important to remember that the tangible output (for example, a specification) is not the goal itself. The goal is to perform the activities, to think, to discuss, to try things out, to make decisions, to document, and whatever else is needed. The tangible output is the way of communicating how the purpose of the process has been fulfilled.

Processes can be described and hence monitored and improved. A process description must always include:



- A definition of the input
- A list of activities—the procedure
- A description of the output

In the basic description of a process, the purpose is implicitly described in the list of activities.

For a more comprehensive and more useful process description the following information could also be included:

- Entry criteria—What must be in place before we can start?
- Purpose—A description of what must be achieved ?
- Role—Who is going to perform the activities?
- Methods, techniques, tools—How exactly are we going to perform the activities?
- Measurements—What metrics are we going to collect for the process?
- Templates—What should the output look like?
- Verification points—Are we on the right track?
- Exit criteria—What do we need to fulfill before we can say that we have finished?

A process description must be operational. It is not supposed to fill pages and pages. It should fit on a single page, maybe even a Web page, with references to more detailed descriptions of methods, techniques, and templates.

2.1.2 Monitoring Processes

It is the responsibility of management in charge of a specific area to know how the pertaining processes are performed. For testing processes it is of course important for the test leader to know how the testing is performed and progressing.

Furthermore, it is important for the people in charge of process improvement to be able to pinpoint which processes should be the target processes for improvement activities and to be able to predict and later determine the effect of process improvement activities.

This is why the description for each process should include the metrics we are interested in for the process, and hence the measurements we are going to collect as the process is being performed.

Metrics and measurements were discussed in Section 1.3, and Section 3.4 discusses how test progress monitoring and control can be performed. In this chapter a few metrics associated with the activities in each of the test processes are listed for inspiration.

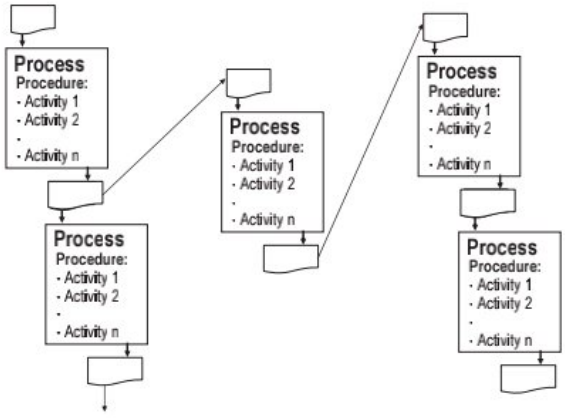
2.1.3 Processes Depend on Each Other

The input to a process must be the output from one or more proceeding process(es)—except perhaps for the very first, where the infamous napkin with the original idea is the input. The output from a process must be the input to one or more other processes—even the final product, which is the input to the maintenance process.

Processes depend on each other.
Output n = Input m

The dependencies between processes can be depicted in a process model, where it is shown how outputs from processes serve as inputs to other processes.

A process model could be in a textual form or it could be graphical, as shown in the following figure. Here, for example, the output from the top-left process serves as input to the top-middle process and to the lower-left process.

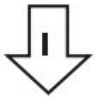


The figure only shows a tiny extract of a process model, so some of the processes deliver input to processes that are not included in the figure.

2.1.4 The Overall Generic Test Process

Testing is a process. The generic test process defined in the ISTQB foundation syllabus can be described like this:

The purpose of the test process is to provide information to assure the quality of the product, decisions, and the processes for a testing assignment.

The inputs on which this process is based are:

- Test strategy
- Project plan
- Master test plan
- Information about how the testing is progressing

The activities are:

- Test planning and control
- Test development
- Test analysis and design
- Test implementation and execution
- Evaluating exit criteria and reporting
- Test closure activities

The output consists of:

- Level test plan
- Test specification in the form of test conditions, test design, test cases, and test procedures and/or test scripts
- Test environment design and specification and actual test environment including test data
- Test logs
- Progress reports
- Test summary report
- Test experience report

The generic test process is applicable for each of the dynamic test levels to be included in the course of the development and maintenance of a product. So the process should be used in testing such as:

- Component testing
- Integration testing
- System testing
- Acceptance testing

The test levels are described in Chapter 1.

Since we apply the view that the concept of testing covers all types of product quality assurance, the generic test process is also applicable to static test (reviews), static analysis (automated static test), and dynamic analysis (run-time analysis of programs). Static testing is described in Chapter 6.

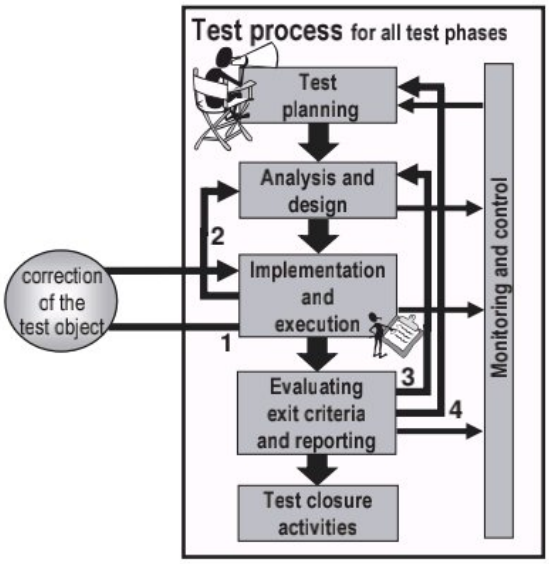
The static test type processes and the level specific test processes depend on each other; and each of them hook into other development processes and support processes. This is described in Chapter 1.

An example of process dependencies is:
The software requirements specification—output from the software requirements specification process—is input to an inspection process and to the system test process.

There are many more dependencies. Some of them are described in the following sections.

The test activities need not be performed in strict sequential order. Test planning and control are constant activities in the sense that they are not just done once in the beginning of the test assignment. Monitoring of the process should be done on an ongoing basis, and controlling and replanning activities performed when the need arises. Sometimes test analysis and design is performed in parallel with test implementation and execution. A model is not a scientific truth; when using a model, even a very well-defined model, we should be open for necessary tailoring to specific situations.

The generic test process is iterative—not a simple straightforward process. It must be foreseen that we'll have to perform the activities more than once in an iterative way before the exit criteria have been fulfilled. The iterations to be foreseen in the test process are shown in the figure here.



Experience shows that in most cases three iterations must be reckoned with as a minimum before the test process can be completed.

The first activity from which an iteration may occur is the test implementation and execution. This is where we detect the failures, when the actual result is different from the expected.

The resulting iterations may be:

1 The defect is in the test object.

A calculation does not give the expected result, and it appears that the algorithm for the calculation has been coded wrongly.

When the defect has been corrected we must retest the software using the test procedure that encountered the failure in the first place. We'll probably also perform some regression testing.

2 The defect is in the test procedure.

A calculation does not give the expected result, but here it appears that the test case was wrong.

The defect must be corrected and the new test case(s) must be executed. This iteration usually goes back to the analysis and design activity.
The second activity from which an iteration may occur is the evaluation of the exit criteria. This is where we find out if the exit criteria are not met.
The resulting iteration in this situation may be:

3 More test cases must be specified to increase coverage, and these must then be executed.

In the checking it turns out that the decision coverage for a component is only 87%. One more test case is designed and when this is executed the coverage reaches 96%.

4 The exit criteria are relaxed or strengthened in the test plan.

The coverage is found to be too small because of an-error handling routine that is very hard to reach. The required coverage for the component is relaxed to 85%.

The generic test process described in detail in the following is primarily aimed for a scripted test where the test is specified before the execution starts.

This does not mean that this test process is not useful for other techniques. Even in exploratory testing where you test a little bit and direct the further test based on the experience gained, you need to plan and control what is going on, to analyze and design (at least in your head), to execute, to report (very important!), and to close the testing.

2.1.5 Other Testing Processes

The test process defined in the ISTQB syllabus is just one example of a possible testing process. Each organization should create its own test process suitable for the specific circumstances in the organization.

A test process may be created from scratch or it may be created as a tailoring of a standard process.

The various process improvement models that exist provide frameworks for processes. Some cover all the process areas in a development organization; others cover the test area in details. Some of the most used process improvement models are discussed in Chapter 8. Two of these are presented here as examples of the framework such process models can provide.

One of the test specific models, *Test Process Improvement Model* (TPI), defines a list of 20 key areas. These cover the total test process and each of them is a potential process in its own right.

The 20 key areas are grouped into four so-called cornerstones as follows:

- Life cycle—Test strategy, life cycle model, moment of involvement
- Techniques—Estimating and planning, test specification techniques, static test techniques, metrics
- Infrastructure—Test tools, test environment, office environment
- Organization—Commitment and motivation, test functions and training, scope of methodology, communication, reporting, defect management, testware management, test process management, evaluating, low-level testing

The TPI model provides inspiration as to which activities could and should be specified for each of these areas when they are being defined as processes in an organization.

Another process model is the Critical Testing Processes (CTP). This model also defines a number of process areas. In this model the process areas are grouped into four classes:

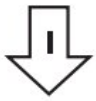
- Plan—Establish context, analyze risks, estimate, plan
- Prepare—Grow and train team, create testware, test environment, and test processes
- Perform—Receive test object(s), execute and log tests
- Perfect—Report bugs, report test results, manage changes

2.2 Test Planning and Control

The purpose of the test planning process is to verify the mission of the testing, to define the objectives of the testing, and to make the necessary decisions to transform the test strategy into an operational plan for the performance of the actual testing task at hand.

The planning must first be done at the overall level resulting in a master test plan. The detailed planning for each test level is based on this master test plan. Identical planning principles apply for the overall planning and the detailed planning.

The purpose of the control part is to ensure that the planned activities are on track by monitoring what is going on and take corrective actions as appropriate.



The inputs on which this process is based are:

- Test strategy
- Master test plan
- Information about how the testing is progressing

The activities are:

- Verify the mission and define the objectives of the testing
- Decide and document how the general test strategy and the project test plan apply to the specific test level: what, how, where, who
- Make decisions and initiate corrective actions as appropriate as the testing progresses

The output consists of:

- Level test plan

2.2.1 Input to Test Planning and Control

The planning of a test level is based on the relevant test strategy, the project plan for the project to which the test assignment belongs, and the master test plan. The contents of these documents, as well as the detailed contents of the level test plan are discussed in Chapter 3.

The level test plan outlines how the strategy is being implemented in the specific test level in the specific project at hand. Basically we can say that the stricter the strategy is and the higher the risk is, the more specific must the level test plan be. Testing and risk is also discussed in Chapter 3.

The test level plan must be consistent with the master test plan. It must also be consolidated with the overall plan for the project in which the testing is a part. This is to ensure that schedules and resources correspond, and that other teams, which interface with the test team in question, are identified.

The decisions to make in the test planning and control process are guided by the expected contents of the test plan. Don't get it wrong: *The decisions are not made for the purpose of writing the plan, but for the purpose of getting agreement and commitment* of all the stakeholders in the test to be performed.

The planning and control of the test are continuous activities. The initial planning will take place first. Information from monitoring what is going on as the testing progresses may cause controlling actions to be taken. These ac-

tions will usually involve new planning and necessary corrections to be made in the plan when it no longer reflects the reality.

2.2.2 Documentation of Test Planning and Control

The tangible output of this process is the level test plan for the testing level to which the process is applied. The structure of the level test plan should be tailored to the organization.

In order not to start from scratch every time it is, however, a good idea to have a template. A template could be based on the IEEE 829 standard. This standard suggests the following contents of a test plan—the words in brackets are the corresponding concepts as defined in this syllabus:

Test plan identifier

1. Introduction (scope, risks, and objectives)
2. Test item(s) (test object(s))
3. Features to be tested
4. Features not to be tested
5. Approach (targets, techniques, templates)
6. Item pass/fail criteria (exit criteria including coverage criteria)
7. Suspension criteria and resumption requirements
8. Test deliverables (work products)
9. Testing tasks (analysis, design, implementation, execution, evaluation, reporting, and closure; all broken down into more detailed activities in an appropriate work break down structure)
10. Environmental needs
11. Responsibilities
12. Staffing and training needs
13. Schedule
14. Risks and contingencies

Test plan approvals

The level test plan produced and maintained in this process is input to all the other detailed test processes. They all have the level test plan as their reference point for information and decisions.

2.2.3 Activities in Test Planning

It cannot be said too often: *Test planning should start as early* as possible. The initial detailed planning for each of the test levels can start as soon as the documentation on which the testing is based has reached a suitable draft level.

The planning of the acceptance testing can start as soon as a draft of the user requirements is available.

Early planning has a number of advantages. It provides, for example, time to do a proper planning job, adequate time to include the stakeholders, early visibility of potential problems, and means of influencing the development plan (e.g., to develop in a sequence that expedites testing).

The test planning activities must first of all aim at setting the scene for the testing assignment for the actors in accordance with the framework. The test planning for a test level must verify the mission and define the objectives—that is the goal or purpose, for the testing assignment. Based on this the more detailed planning can take place.

2.2.3.1 Defining Test Object and Test Basis

The object of the testing depends on the test level as described in Chapter 1. Whatever the test object is, the expectations we have for it, and therefore what we are going to test the fulfillment of, should be described in the test basis.

The test planning must identify the test basis and define what it is we are going to test in relation to this. This includes determination of the coverage to achieve for the appropriate coverage item(s). The expected coverage must be documented in the level test plan as (part of) the completion criteria. The coverage items depend on the test basis.

Examples of the most common test basis and corresponding coverage items are listed in the following table.

Test level	Test basis	Coverage items
Component testing	◗ Requirements ◗ Detailed design ◗ Code	◗ Statements ◗ Decisions ◗ Conditions
Component integration testing	◗ Architectural design	◗ Internal interfaces ◗ Individual parameters ◗ Invariants
System testing	◗ Software requirements specification	◗ Requirements: - functional - nonfunctional
System integration testing	◗ Product design	◗ External interfaces ◗ Individual parameters ◗ Invariants
Acceptance testing	◗ User requirements specification ◗ User manual	◗ Requirements expressed as - use cases - business scenarios

Static test	• Documents the static test is based on	• Pages • Requirements • Test cases

Standards, both internal and external to the organization, may also be used as the test basis.

2.2.3.2 Defining the Approach

The test approach must be based on the strategy for the test at hand. This section expands the approach and makes it operational.

The approach must at least cover:

- The test methods and test techniques to use
- The structure of the test specification to be produced and used
- The tools to be used
- The interface with configuration management
- Measurements to collect
- Important constraints, such as availability or "fixed" deadline

for the testing we are planning for.

First of all, the test object determines the *method:*

- If the test object is something that can be read or looked at, the method is static test—the specific choice of static test type(s) depends on the criticality of the object.
- If the test object is executable software, the method is dynamic test.

For each of the dynamic test types a number of *test case design techniques* may be used. The test case design techniques are discussed in detail in Chapter 4. The choice of test techniques is dependent on the test object, the risks to be mitigated, the knowledge of the software under testing, and the nature of the basis document(s). The higher the risk, the more specific should the recommendation for the test case design techniques to use be, and the more thorough should the recommended test case design techniques be.

The *structure of the test specification* must be outlined here. Test specifications may be structured in many ways—for example, according to the structure suggested in IEEE 829. This is described in Section 2.3.2.

The *usage of tools* must also be described in the approach. Tools are described in Chapter 10. *The strategy for the tool usage must be adhered to.*

The interface with *configuration management* covers:

- How to identify and store the configuration items we produce in the test process
- How to get the configuration items we need (for example, design specifications, source code, and requirements specifications)
- How to handle traceability
- How to register and handle incidents

A reference to descriptions in the configuration management system, should suffice here, but we are not always that lucky. If no descriptions exist we must make them—and share them with those responsible for configuration management.

The *measurements* to be collected are used for monitoring and control of the progress of the testing. We must outline what and how to measure in the approach. Measurements are discussed in detail in Sections 1.3 and 3.4.

2.2.3.3 Defining the Completion Criteria

The completion criteria are what we use to determine if we can stop the testing or if we have to go on to reach the objective of the testing.

The completion criteria are derived from the strategy and should be based on a risk analysis; the higher the risk, the stricter the completion criteria; the lower the risk the less demanding and specific the completion criteria.

It is important to decide up front which completion criteria should be fulfilled before the test may be stopped.

The completion criteria guide the specification of the test and the selection of test case design techniques. These techniques are exploited to provide the test cases that satisfy the completion criteria. Test case design techniques are discussed in detail in Chapters 4 and 5.

The most appropriate completion criteria vary from test level to test level. Completion criteria for the test may be specified as follows:

- Specified coverage has been achieved
- Specified number of failures found per test effort has been achieved
- No known serious faults
- The benefits of the system are bigger than known problems
- (The time has run out)

The last one is not an official completion criterion and should never be used as such; it is nonetheless often encountered in real life!

Coverage is a very often used measurement and completion criteria in testing. Test coverage is the degree, expressed as a percentage, to which the coverage items have been exercised by a test.

The above mentioned completion criteria may be combined and the completion criteria for a test be defined as a combination of more individual completion criteria.

Examples of combinations of completion criteria for each of the test levels may be:

- Component testing
  - 100% statement coverage
  - 95% decision coverage
  - No known faults
- Integration testing (both for components and systems)
  - 90% parameter coverage
  - 60% interface coverage
  - No known faults
- System testing
  - 90% requirement coverage
  - 100% equivalence class coverage for specific requirements
  - No known failures of criticality 1 or 2
  - Stable number of failures per test hour for more than 20 test hours
- Acceptance testing
  - 100% business procedure coverage
  - No known failures of criticality 1

2.2.3.4 Defining Work Products and Their Relationships

The number of deliverables, their characteristics, and estimates of their sizes must be defined, not least because this is used as input for the detailed estimation and scheduling of all the test activities, but also because the precision of what is going to be delivered sets stakeholders' expectations.

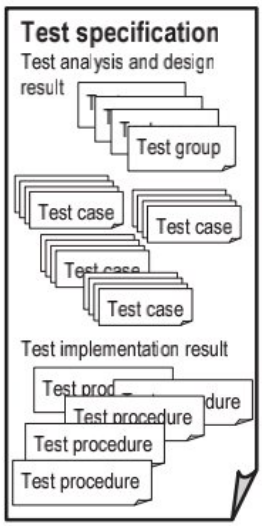
Typical deliveries or work products from a test level are:

- Level test plan(s)
- Test specification(s)
- Test environment(s)
- Test logs and journals
- Test reports

The level test plan is the plan being specified in this process.

The test specification is a collective term for the result of the test design and implementation activities. This is the most complicated of the work products. It is important that the *structure of the test specification* is outlined in the level test plan, so that its complexity is understood and taken into consideration when the effort is estimated, and also to guide the work in the subsequent activities.

Test specifications may be structured in many ways. Each organization must figure out which structure is the most suitable for them. No matter the structure the test specification could be held in one document or in several separate documents; the physical distribution of the information is not important, but the actual contents are.



The structure shown and explained here is based on the structure suggested in IEEE 829. A full test specification may consists of:

- A *test design* consisting of a number of test groups (or designs) with test conditions and high-level test cases derived from the basis documentation. The designs will typically reflect the structure of the test basis documentation. The relationships between the elements in the basis documentation and the high-level test cases may well be quite complicated, often including even many-to-many relationships.
- A number of *low-level test cases* extracted from the high-level test cases and being made explicit with precise input and output specifications .
- A number of test procedures each encompassing a number of test cases to be executed in sequence. The relationships between high-level test cases and test procedures may also be complicated and include many-to-many relationships.

This structure is applicable to test specifications at all test levels, for example, for:

- Component testing
- Integration testing
- System testing
- Acceptance testing

The detailed contents of the test specification are discussed in Section 2.3.3.

2.2.3.5 Scoping the Test Effort

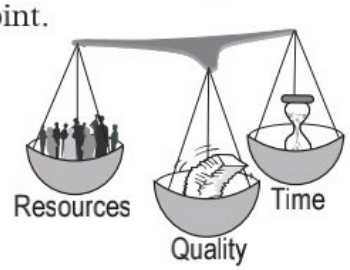
The definition of exhaustive testing is: test case design technique in which the test case suites comprise all combinations of input values and preconditions for component variables. No matter how much we as testers would like to do the ultimate good job, *exhaustive testing is a utopian goal.*

We do not have unlimited time and money; in fact we rarely have enough to obtain the quality of the testing we would like. It would in almost all cases take an enormous amount of resources in terms of time and money to test exhaustively and is therefore usually not worth it.

We have three mutually dependent parameters that we as testers need to balance. In fact, for everything we do, we need to balance these parameters, but here we'll look at them from a tester's view point.

The parameters are:

Time: The available calendar time
Resources: The available people and tools
Quality: The quality of the testing



These parameters form what we call the quality triangle, that is the triangle for the quality of work we can deliver.

In a particular project we need to initially achieve a balance between the time and resources spent on testing and the quality of the testing we want.

The basic principle of the quality triangle is: *It is not possible to change one of the parameters and leave the other two unchanged—and still be in balance!*

The time and the resources are fairly easy concepts to understand. Testing takes time and costs resources. The quality of the testing is more difficult to assess. The easiest way to measure that quality is to measure the test coverage. The test coverage is the percentage of what we set out to test (e.g., statements) that we have actually been able to cover with our test effort.

Test coverage is a measure for the quality of the test.

When we perform the test planning we need to look further ahead than the horizon of testing. Important factors could cause one of the parameters in the quality triangle for testing to be fixed.

It may, for example, be necessary:

- To fix a release date for economical or marketing reasons if the product must be presented at the yearly sales exhibition for the particular type of product
- To keep a given price, especially in fixed price projects
- To obtain a specific level of quality, for example in safety critical products

Everything needs to be balanced. The time and cost of testing to enhance the quality must be balanced with the cost of missing a deadline or having remaining defects in the product when it goes on the market.

Work Breakdown Structure

One of the things on which the test planning is based, is a list of all the tasks to be performed. This list should be in the form of a work breakdown structure of the test process at hand. If we use the test process defined here the overall tasks are planning, monitoring, control, analysis, design, implementation, execution, evaluation, reporting, and closure, all broken down into more detailed activities in an appropriate work breakdown structure.

The tasks, together with resources and responsibilities, are input items to the test schedule.

A list and a description of every single task must therefore be produced. If a task is not mentioned here it will probably not get done. *Be conscientious: remember to remember EVERYTHING!* Be as detailed as necessary to get a precise estimate. A rule of thumb is to aim at a break down of activities to tasks that can be done in no more than about 30 to 40 hours.

All the activities in the test process must be included in the task list. Do not forget to include the test management activities like planning, monitoring, and control. Also remember that the estimation and scheduling takes time—these activities must be included as well.

It is important here to remember that the test process is iterative. This must of course be taken into account during the estimation, but it will facilitate the estimation if iterations of activities are explicitly mentioned in the task list.

Defining Test Roles

A (software test) project is like a play in which all roles must be filled in order for the play to be performed. Some roles are big, some are small, but they are all important for the whole.

Real people must fill the roles. Real people vary; they have different personalities, a fact of life that it is almost impossible to change. Technical skills you can learn, but your personality is to a large extent fixed when you reach adulthood. Different people fill different roles in different ways, and the differences between people may be used to the advantage of everybody, if the basics of team roles are known. This is discussed in detail in Chapter 10.

It is of great importance in the general understanding of the work to be done that the roles are described. Processes and procedures may be described thoroughly, but only when the activities and tasks are connected to roles and thereby to real people do they become really meaningful.

The roles to handle the testing tasks may be:

- Test leader (manager or responsible)
- Test analyst/designer
- Test executer
- Reviewer/inspector
- Domain expert
- Test environment responsible
- (Test)tool responsible

Test teams are formed by all these roles. We need different teams depending on which test phase we are working in, but the principles are the same:

- *All relevant roles must be present and filled in the team*
- A role can be filled by one person or more people, depending on the size of the testing assignment at hand
- One person can fill one role or more roles, again depending on the size (but keep in mind that less than 25% time for a role = 0% in real life)

The roles are assigned to organizational units and subsequently to named people. The necessary staff to fulfill the roles and take on the responsibilities must be determined.

The roles each require a number of specific skills. If these skills are not available in the people you have at your disposal, you must describe any training needs here. The training should then be part of the activities to put in the schedule.

Producing the Schedule

In scheduling the tasks, the staffing and the estimates are brought together and transformed into a schedule. Risk analysis may be used to prioritize the testing for the scheduling: the higher the risk, the more time for testing and the earlier the scheduled start of the testing task.

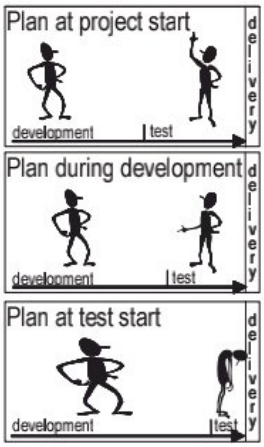
The result of this is a schedule that shows precisely who should do what at which point in time and for how long.

A framework for the resources and the schedule must be obtained from the overall project plan, and the result of the test scheduling must be reconciled with the project plan.

Estimations for all the tasks are input to the scheduling. Once the tasks are estimated they can be fitted into the project time line. Test estimation is discussed in detail in Section 3.3.

The schedule is also based on the actual people performing the tasks, the people's efficiency, and their availability.

2.2.4 Activities in Test Control



As the testing progresses the control part of test management is about staying in control and making necessary corrections to the plan when it no longer reflects the reality.

Measurements are collected in the test monitoring activities for all the detailed activities in the test processes, and these measurements are analyzed to understand and follow the actual progress of the planned test activities and the resulting coverage. Decisions must be made if things are deviating significantly from the plan, and corrective actions may be necessary.

The testing often gets pressed for time, since it is the last activity before the product is released. When development is delayed it is tempting to shorten the test to be able to keep the release date.

But if our testing time is cut, we have to change at least one other parameter in the quality triangle; anything else is impossible. It is important to point this out to management. It is irresponsible if for example the consequences on resources and/or testing quality of a time cut are not made clear. If it looks as if we are going to end up in the all too familiar situation illustrated here, we have to take precautions.

There is more about test monitoring and control in Section 3.4

2.2.5 Metrics for Test Planning and Control

Metrics to be defined for the monitoring and control of the test planning and control activities themselves may include:

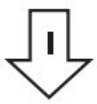
- Number of tasks commenced over time
- Task completion percentage over time
- Number of tasks completed over time
- Time spent on each task over time

This will of course have to be compared to the estimates and schedule of the test planning and control activities.

2.3 Test Analysis and Design

The purpose of the test analysis and design activities is to produce test designs with test conditions and tests cases and the necessary test environment based on the test basis and the test goals and approach outlined in the test plan.

The inputs on which this process is based are:



- Level test plan
- Basis documentation

The activities are:
- Analysis of basis documentation
- Design of high-level test cases and test environment

The output consists of:
- Test design
- Test environment design and specification

2.3.1 Input to Test Analysis and Design

The input from the level test plan that we need for this process is:

- Test objectives
- Scheduling and staffing for the activities
- Definition of test object(s)
- Approach—especially test case design techniques to use and structure and contents of the test specification
- Completion criteria, not least required coverage
- Deliverables

We of course also need the test basis—that is, the material we are going to test the test object against.

2.3.2 Documentation of Test Analysis and Design

The result of the test analysis and design should be documented in the test specification. This document or series of documents encompasses

- The test designs—also called test groups
- The test cases—many test cases per test design
- Test procedures—often many-to-many relationship with test cases

The overall structure of the test specification is defined in the level test plan. The detailed structure is discussed below.

The test specification documentation is created to *document the decisions* made during the test development and to *facilitate the test execution*.

2.3.3 Activities in Test Analysis and Design

The idea in structured testing is that the test is specified before the execution. The test specification activity can already start when the basis documentation is under preparation.

The test specification aims at designing tests that provide the largest possible coverage to meet the coverage demands in the test plan. This is where test case design techniques are a great help.

The work with the specification of the test groups, the test conditions, the test cases, and the test procedures are highly iterative.

A side effect of the analysis is that we get an extra review of the basis documentation. Don't forget to feed the findings back through the correct channels, especially if the basis documentation isn't testable.

2.3.3.1 Defining Test Designs

In test design the testing task is broken into a number of test design or test groups. This makes the test development easier to cope with, especially for the higher test levels. Test groups may also be known as test topics or test areas.

A *test design* or test group specification should have the following contents according to IEEE 829:

Test design specification identifier

1. Features to be tested (test conditions)
2. Approach refinement
3. List of high-level test cases
4. List of expected test procedures
5. Feature pass/fail criteria

Test design specification approvals

The groups and the procedures must be uniquely identified. The number of test groups we can define depends on the test level and the nature, size, and architecture of the test object:

- In component testing we usually have one test group per component
- For integration testing there are usually a few groups per interface
- For system and acceptance testing we typically have many test groups

A few examples of useful test groups defined for a system test are:

- Start and stop of the system
- Functionality x
- Nonfunctional attribute xx
- Error situations

It should be noted that it is not very common to document the test design as thoroughly as described here. Often a list of groups with a short purpose description and list of the test procedures for each are sufficient.

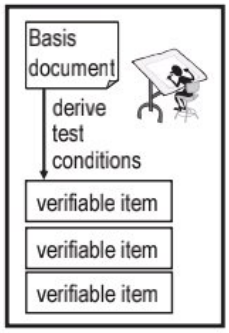
Test group: 2 (2) Handling member information
The purpose of this test group is to test that the member information can be created and maintained.

Test procedure: 2.1 (10) Creating new member
Test procedure: 2.2 (14) Changing personal information
Test procedure: 2.3 (11) Changing bonus point information
Test procedure: 2.4 (13) Deleting member

The unique identification is the number in brackets, for example (10). The number before the unique identifier is the sorting order to ensure that the groups and procedures are presented in a logical order independently of the unique number, for example 2.1. The "disorder" of the unique identification is a sign of the iterative way in which they have been designed.

2.3.3.2 Identification of Test Conditions

The features to be tested mentioned in the test design can be expressed as test conditions or test requirements. A test condition is a verifiable item or element.



The nature of a test condition depends on the nature of the test basis documentation. It may for example be a function, a transaction, a feature, a requirement, or a structural element like an interface parameter or a statement in the code.

The test conditions are based on or identical to our coverage items. They are the items we are covering when we test the test object.

We cannot expect to be able to cover 100% of all the relevant coverage items for our test. This is where we take the completion criteria into account in our specification work.

The completion criteria often include the percentage of the coverage items we must cover, called the coverage. We select the test conditions to get the highest coverage. Prioritization criteria identified in the risk analysis and test planning may be applied in the analysis activity to pick out the most important coverage items if we cannot cover them all.

The completion criteria for a component test could include a demand for 85% decision coverage.

If we are lucky the test conditions are clearly specified and identifiable in the test basis documentation, but in many cases it can be quite difficult. The earlier testers have been involved in the project, the easier this task usually is.

The documentation of a test condition must at least include:

- Unique identification
- Description
- Reference to test basis documentation, if not taken from there directly

The example here is based on the EuroBonus scheme of StarAlliance. This short description is taken from the SAS Web site:
There are 3 member levels: Basis, Silver, Gold.
Your member level is determined by the number of Basis Points you earn within your personal 12-months period. You will automatically be upgraded to Silver Member if you earn 20.000 Basis Points during your earning period.
If you earn 50.000 Basis Points in the period, you become a Gold Member. The earning period runs from the first day in the joining month and 12 months forward.

Some of the test conditions that which can be extracted from this are:

1) When the sum of basis points is less than 20.000, the member status is Basis.
2) When the sum of basis points is equal to or greater than 20.000, the member level is set to Silver.
3) When the sum of basis points is equal to or greater than 50.000, the member level is set to Gold.
There are many more—and just as many questions to be posed!

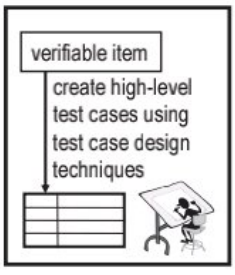
Only if the test conditions are not clearly defined in the basis documentation do we have to document them ourselves. If we do so we must *get the test conditions reviewed* and approved by the stakeholders.

2.3.3.3 Creation of Test Cases

Based on the test conditions, we can now produce our first high-level test cases and subsequently low-level test cases.

A high-level test case is a test case without specific values for input data and expected results, but with logical operators or other means of defining what to test in general terms.

The test cases we design should strike the best possible balance between being:



- Effective: Have a reasonable probability of detecting errors
- Exemplary: Be practical and have a low redundancy
- Economic: Have a reasonable development cost and return on investment
- Evolvable: Be flexible, structured, and maintainable

The test case design techniques make it possible to create test cases that satisfy these demands.

The test techniques help us identify the input values for the test cases.
The techniques cannot supply the expected result.

We use appropriate test case design technique(s) as specified in the test level plan to create the high-level test cases. Test case design techniques are discussed in Chapter 4.

The documentation of a *test case* at this stage must at least include:

- Unique identification
- Description
- References to test condition(s) on which the test case is based and to test design(s) to which the test case belongs

There may well be many-to-many relationships between test conditions and high-level test cases and/or between high-level test cases and test designs.

Even though IEEE is quite specific in its requirements for the test specification it is not very often that test conditions and high-level test cases are officially documented. They are usually sketched out during the analysis and design work. Only the test designs and their procedures and low-level test cases are kept in the test specification. The decision about how much documentation of test conditions and high-level test cases to keep must be based on the strategy and the risks involved.

From the test conditions in the earlier example we can design the following high-level test cases using the equivalence partitioning technique:
HTC 1) Check that a negative sum of basis points is not allowed.
HTC 2) Check that a sum of basis points of less than 20.000 will give a membership level basis.
HTC 3) Check that a sum of basis points of more than 20.000 and less than 50.000 will give a membership level silver.
HTC 4) Check that a sum of Basis Points of more than 50.000 will give a membership level gold.

The analysis of the basis documentation will also reveal requirements concerning the test environment, not least the required test data. The test environment should be specified to a sufficient level of details for it to be set up correctly; and it should be specified as early as possible for it to be ready when we need it. Test environment requirements are discussed later.

From the high-level test cases we go on to define the low-level test cases. It is not always possible to execute all the test cases we have identified; the actual test cases to be executed must be selected based on the risk analysis.

A low-level test case is a test case with specific values defined for both input and expected result.

The documentation of a *low-level test case* must at least include:

- Unique identification
- Execution preconditions
- Inputs: data and actions
- Expected results including postconditions
- Reference(s) to test conditions and/or directly to basis documentation

One low-level test case created from the list of these high-level test cases could be:

ID	Precondition	Input	Expected result	Postcondition
15.2	The current sum of basis points for Mrs. Hass is 14.300 The system is ready for entry of newly earned basis points for Mrs. Hass.	Enter 6.500 Press [OK]	The sum is shown as 20.800 The member status is shown as silver	The system is ready for a new member to be chosen.

The expected result must be determined from the basis documentation where the expectations for the coverage items are described. *The expected result must never, ever be derived from the code!*

The expected results should be provided in full, including not only visible outputs but also the final state of the software under testing and its environment. This may cover such factors as changed user interface, changed stored data, and printed reports.

We may, for example, have the following test cases, where the first gives a visible output and the second does not give a visible output, but makes a new form current.

Case	Input	Expected result
1.	Enter "2" in the field "Number of journeys:"	Value in the field "Total points:" is the value in field "Points per journey:" x 2.
2.	Try to enter "10" in the field "Number of journeys:"	Value in the field "Total points:" is unchanged. The error message pop-up is current and showing error message no. 314.

In some situations the expectations may not be formally specified. Therefore it is sometimes necessary to identify alternative sources, such as technical and/or business knowledge. RAD is a particular example of where the requirements may not be formally specified.

If it turns out that it is not possible to identify what to test against, you must *never, ever just guess or assume*. Nothing to test against entails no test!

Sometimes it can be difficult to determine the expected result from the basis documentation. In such cases an oracle may be used. Oracles are discussed under tools in Section 9.3.2.

It cannot be pointed out strongly enough that if you guess about what to test and go ahead with the test specification based on your assumptions and guesses, *you are wasting everybody's time*. The chance of your getting it right is not high.

You also prevent your organization from getting better, because the people responsible for the source documentation will never know that they could easily do a better job. Go and talk to the people responsible for the source documentation. Point out what you need to be able to test. Make suggestions based on your test experience. Use some of the methods from test techniques to express the expectations, for example decision tables. Help make the source documentation better.

2.3.4 Requirements

This book is about testing, not requirements. A short introduction to requirements is, however, given in this section. The purpose of this is to make testers understand requirements better, and equip them to take part in the work with the requirements and to express test-related requirements for the requirements produced for a product.

All product development starts with the requirements. The higher level testing is done directly against requirements. The lower level testing is done against design that is based on the requirements. All testing is hence based on the requirements.

2.3.4.1 Requirement Levels

Requirements should exist at different levels, for example:

- Business requirements
- User requirements
- System requirements

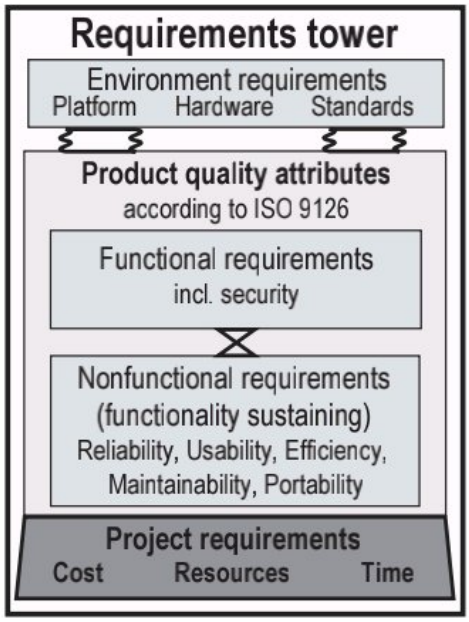
Requirements are rooted in or belong to different stakeholders. Different stakeholders speak different "languages" and the requirements must be expressed to allow the appropriate stakeholders to understand, approve, and use them.

The organization and top management "speak" money—they express business requirements. Business requirements may be tested, but most often they are not tested explicitly.

The users speak "support of my work procedures"—they express user requirements. User requirements are tested in the acceptance testing.

Following a possible product design, where the product is split up in, for example, a software system and a hardware system, we must express the system requirements. The software requirements are for the software developers and testers, and they are tested in the system testing.

2.3.4.2 Requirement Types



The requirement specification at each level must cover all types of requirements.

The most obvious requirements type is functional. No functionality entails no system. But as important as it may be, the functionality is not enough.

We must have some requirements expressing how the functionality should behave and present itself. These requirements are usually known as nonfunctional requirements. We could also call them functionality-supporting requirements. These requirements are discussed in detail in Chapter 5.

The functional and nonfunctional requirements together form the product quality requirements.

On top of this we may have environment requirements. These are requirements that are given and cannot be discussed. They can come both from inside and outside of the organization and can be derived from standards or other given circumstances. Environment requirements may, for example, define the browser(s) that a Web system must be able to work on, or a specific standard to be complied with.

To make the requirements tower balance we need to have project requirements (or constraints) to carry the other requirements. These are cost-, resources-, and time-related, and the worry of the project management.

2.3.4.3 Requirement Styles

Requirements can be expressed in many ways. Typical styles are:

- Statements
- Tasks
- Models
- Tables

The most common style is the statement style. Here each requirement is expressed as a single (or very few) sentences in natural language. Some rules or recommendations should be observed when expressing requirements in statements:

- Start with: "The product shall …"—to keep focus on the product or system
- Avoid synonyms—stick to a defined vocabulary
- Avoid subjective words (useful, high, easy)—requirements must be testable!
- Avoid generalities like "etc." "and so on"—this is impolite; think the issue through
- Be aware of "and" and "or"—is this really two or more requirements?

To make statement requirements more precise and testable we can use metrics and include information such as the scale to use, the way to measure, the target, and maybe acceptable limits. This is especially important for non-functional requirements!

Examples of such requirements (with unique numbers) are:
[56] The maximum response time for showing the results of the calculation described in requirements 65 shall be 5 milliseconds in 95% of at least 50 measurements made with 10 simultaneous users on the system.

[UR.73] It shall take a representative user (a registered nurse) no more than 30 minutes to perform the task described in use case 134 the first time.

A *task* is a series of actions to achieve a goal. Task styles may be stories, scenarios, task lists, or use cases. Requirements expressed in these ways are easy to understand, and they are typically used to express user requirements. They are easy to derive high-level test cases and procedures from.

A *model* is a small representation of an existing or planned object. Model styles may be domain models, prototypes, data models, or state machines.

A *table* is a compact collection and arrangement of related information. Tables may be used for parameter values, decision rules, or details for models.

The styles should be mixed within each of the requirement specifications so that the most appropriate style is always chosen for a requirement.

The collection of requirements for each level documented in the requirement specification is in fact a model of the product or the system. This model is the one the test is based on.

2.3.5 Traceability

References are an important part of the information to be documented in the test specification. A few words are needed about these.

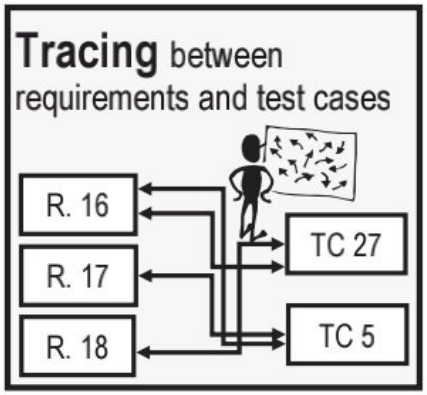
There are two sets of references:

- References between test specification elements
- References from test specification elements to basis documentation

The first set of references describes the structure of the elements in the test specification. These may be quite complex with, for example, test cases belonging to more test procedures and more test groups.

The references to the basis documentation enable traceability between what we are testing and how we are testing it. This is very important information. Ultimately traceability should be possible between test cases and coverage items in the basis documentation.

Traces should be two-way.



You should be able to see the traces from the test cases to the covered coverage items. This will help you to identify if there are test cases that do not trace to any coverage item—in which case the test case is superfluous and should be removed (or maybe a specification like a requirement or two should be added!). This "backward" trace is also very helpful if you need to identify which coverage item(s) a test case is covering, for example, if the execution of the test case provokes a failure.

You should also be able to see the traces from the coverage items to the test cases. This can be used to show if a coverage item has no trace, and hence is not covered by a test case (yet!). This "forward" trace will also make it possible to quickly identify the test case(s) that may be affected if a coverage item, say a requirement, is changed.

If the coverage items and the test cases are uniquely identified, preferably by a number, it is easy to register and use the trace information.

Instead of writing the trace(s) to the coverage item(s) for each test case, it is a good idea to collect the trace information in trace tables. This can be done in the typical office automation system, such as in a Word table, Excel, or (best) a database.

The example on the opposite page is an extract of two tables, showing the "forward" and the "backward" traces between test cases and requirements, respectively.

Requirements to Test Cases	From Test Cases to Requirements
9.1.1.5 (8)	**3.1 (7)**
2.5 (45)	10.2.1.3.a (74)
9.1.2.1.a (14)	10.6.2.7.a (123)
5.1 (10)	10.6.2.7.b (124)
9.1.2.1.b (15)	10.6.2.10.c (131)
5.3 (13)	**3.2 (36)**
5.4 (14)	10.5.1.1 (98)
5.5 (12)	10.5.1.3 (100)

2.3.6 Metrics for Analysis and Design

Metrics to be defined for the monitoring and control of the test analysis and design activities may include:

- Number of specified test conditions and high-level requirements over time
- Coverage achieved in the specification (for example, for code structures, requirements, risks, business processes), over time
- Number of defects found during analysis and design
- Other tasks commenced and completed over time, for example, in connection with test environment specifications
- Time spent on each task over time

This will, of course, have to be compared to the estimates and schedule of the test analysis and design activities.

2.4 Test Implementation and Execution

The purpose of the test implementation is to organize the test cases in procedures and/or scripts and to perform the physical test in the correct environment.

The inputs on which this process is based are:

- Level test plan
- Test conditions and test design
- Other relevant documents
- The test object

The activities are:

- Organizing test procedures
- Design and verify the test environment
- Execute the tests

This is the first place from which iterations may occur

- Record the testing
- Check the test results

The output consists of:

- Test specification
- Test environment
- Test logs
- Incident reports
- Tested test object

2.4.1 Input to Test Implementation and Execution

The input from the level test plan that we need for this process is:

- Scheduling and staffing for the activities
- Definition of the test object(s)
- Specification of test environment
- Entry criteria for the test execution
- Exit criteria, including coverage criteria

From the test analysis and design process we need the test specification in its current state.

We might need other documentation, for example, a user manual, documentation of completion of preceding test work, and logging sheets. For the actual execution of the test we obviously need the test object.

2.4.2 Documentation of Test Implementation and Execution

The test specification is finished in this process where the test procedures are laid out. During this work the requirements concerning the test environment are finalized.

The test environment must be established before the test execution may start. In some cases the test environment is explicitly documented.

The test execution is documented in test logs. When failures occur these should be documented in incident reports.

2.4.3 Activities in Test Implementation and Execution

2.4.3.1 Organizing Test Procedures

The low-level test cases should now be organized and assembled in test procedures and/or test scripts.

The term "procedure" is mostly used when they are prepared for manual test execution, while the term "script" is mostly used for automatically executable procedures.

The *degree of detail in the procedures depends on who will be executing the test.* They should therefore always be written with the intended audience in mind. Experienced testers and/or people with domain knowledge and knowledge about how the system works will need far less details than "ignorant" testers.

What we need to specify here is the actual sequence in which the test cases should be executed.

The documentation of a *test procedure* must at least include:

- Unique identification
- Description
- References to high-level test cases and/or to test conditions and/or directly to basis documentation to be covered by the procedure
- An explicit description of the preconditions to be fulfilled before the actual test execution can start
- Included low-level test cases

Test procedures may be organized in such a way that the execution of one test procedure sets up the prerequisites for the following. It must, however, also be possible to execute a test procedure in isolation for the purpose of confirmation testing and regression testing. The prerequisites for a test procedure must therefore always be described explicitly.

Test procedures may be hierarchical, that is "call others," for example, generic test cases.

The test groups and the specification of their test procedures must be revisited to ensure that they are organized to give a natural flow in the test execution. Remember that the production of the test specification is an iterative process. We need to keep on designing and organizing test cases, test procedures, and test groups until everything falls into place and we think we have achieved the required coverage.

The organization in test procedures could be looked at as the execution schedule. It could be fixed, but it could also be dynamic. For specific purposes, especially for regression testing, some of the test procedures may be selected and reorganized in other execution schedules that fit the specific purpose.

A test procedure should not include too many or too few test cases—a maximum of 20 test cases and a minimum of 2–4 test cases is a good rule of thumb.

The test procedure may also include facilities for logging the actual execution of the procedure.

There are many ways to lay out the specification of test procedures and test cases. It is a good idea to set up a template in the organization.

Here is an example of a template for a test procedure. The procedure heading contains fields for the required information and fields to allow the procedure to be used for logging during test execution. The template for the cases contains unique numbering of the case (within the procedure), input and expected result, and a column for registration of the actual result to be used for logging during execution.
Test procedure: n.n (n)

Test procedure:			
Purpose: This test procedure tests ... Traces:			
Prerequisites: Set up ...			
Expected duration: *x* minutes			
Execution information			
Test date and time:		**Initials:**	
Test object identification:		**Result:**	
Case	**Input**	**Expected result**	**Actual result**
1.			
2.			

Note that the template indicates a unique identification of the test procedure (n), and a number indicating its position among all the other test procedures (n.n).

To facilitate estimation the test designer is required to provide an estimate of the execution time for manual execution of the test procedure.

Quality Assurance of the Test Specification

Before the test specification is used in the test execution it should be reviewed. The review should ensure that the test specification is correct with respect to the test basis, including any standards, that it is complete with respect to the required coverage, and that it can be used by those who are going to execute the test.

Apart from the obvious benefits of having the test specification reviewed, it also has some psychological benefits. Usually we as testers review and test the work products of the analysts and developers, and we deliver feedback in

the form of verbal or written review reporting and incident reports.

This may make us seem as those who are always the bearers of bad news and ones who never make any mistakes ourselves. Getting the analysts and developers to review our work will reverse those roles; it will make us learn what it is like to receive feedback, and it will make the analysts and developers learn what it is like to deliver feedback and learn that even testers make mistakes!

The review may be guided by a checklist, of which a very small example is shown here:

- Is the test specification clear and easily understood?
- Is the test structure compatible with automated test?
- Is it easy to maintain?
- Is it easy for others to perform a technical review?

2.4.3.2 Test Environment Specification and Testing

The test environment is a necessary prerequisite for the test execution—without a proper environment the test is either not executable at all or the results will be open to doubt.

The environment is first outlined in the test plan based on the strategy. The test plan also describes by whom and when the test environment is to be created and maintained. Some additional requirements for the environment may be specified in the test specification in the form of prerequisites for the test procedures, and especially for test data. The exact requirements for test data needed to execute test procedures may only be determined quite close to the actual execution. It is very important that planning and facilities for setting up specific test data are made well in advance of the execution.

The description of the test environment must be as specific as possible in order to get the right test environment established at the right time (and at the right cost). Beware: *The setting up of the test environment is often a bottleneck* in the test execution process, mostly because it is insufficiently described, underestimated, and/or not taken seriously enough. Either the environment is not established in time for the actual test execution to begin and/or it is not established according to the specifications. If the test environment is not ready when the test object is ready for the test to be executed, *it jeopardizes the test schedule*. If it is not correct, *it jeopardizes the trustworthiness of the test*.

The descriptions of the test environment must cover:

- Hardware—to run on and/or to interface with
- Software—on the test platform and other applications
- Peripherals (printers including correct paper, fax, CD reader/burner)

- Network—provider agreements, access, hardware, and software
- Tools and utilities
- Data—actual test data, anonymization, security, and rollback facilities
- Other aspects—security, load patterns, timing, and availability
- Physical environment (room, furniture, conditions)
- Communication (phones, Internet, paper forms, paper, word processor)
- Sundry (paper, pencils, coffee, candy, fruit, water)

Problems with the test environment may force testing to be executed in other less suitable environments. The testing could be executed in inappropriate competition with other teams and projects. If we test in the development environment, test results can be unpredictable for inexplicable reasons due to the instability of this environment. In the worst case, testing is executed in the production environment where the risk to the business can be significant.

The specific requirements for the test environment differ from test level to test level. The test environment must, at least for the higher levels of testing, be as realistic as possible, that is it should reflect the future production environment.

The need for the environment to reflect the production environment is not as pronounced for the lower test levels. In component testing and integration testing the specification must, however, include requirements concerning any drivers and stubs.

It may in some cases be *too expensive, dangerous, or time-consuming* to establish such a test environment. If this is the case the test may be un-executable and other test methods, like inspection of the code, may be used to verify the product.

As the testers we are, we have to verify that the test environment is complete according to the specifications and that it works correctly before we start to execute our test procedures. We must ensure that the test results we get are valid, that is if a test passes, it is because the test object is correct, and if it fails it is because the test object, and not the test environment, has a defect—and vice versa.

2.4.3.3 Checking Execution Entry Criteria

Even though we are eager to start the test execution we should not be tempted to make a false start. We need to make sure that the execution entry criteria are fulfilled.

If the test object has not passed the entry criteria defined for it, do not start the test execution. You will waste your time, and you risk teaching the developers or your fellow testers that they don't need to take the entry criteria seriously.

We of course also need to have the people taking part in the test execution available, as specified in the test plan. The test executors must be appropriately trained, and any stakeholders needed, for example, customers to witness the execution, must be present and briefed.

Efficient and timely execution of the tests is dependent on the support processes being in place. It is particularly important that the configuration management is working well, because of the interfaces between the testing process and the configuration management process, including:

- The ability to get the correct version of the test object, the test specification, and/or the ability to get the correct versions of any other necessary material
- The ability to be able to report the failures and other incidents found during the testing
- The ability to follow the progress of the failures and plan any necessary confirmation testing and regression testing
- The ability to register approval of successful removal of failures

Support processes are discussed in Chapter 1.

2.4.3.4 Test Execution

The execution of the tests is what everybody has been waiting for: the moment of truth!

In structured testing, as we have discussed earlier, in principle all the testers have to do during test execution is to follow the test specification and register all incidents on the way. If the execution is done by a tool, this is exactly what will happen.

We have taken great care in writing the test procedures, and it is important to follow them. There are several reasons for this:

- We need to be able to trust that the specified testing has actually been executed.
- We need to be able to collect actual time spent and compare it with the estimates to improve our estimation techniques.
- We need to be able to compare the progress with the plan.
- We need to be able to repeat the tests exactly as they were executed before for the sake of confirmation testing and regression testing.
- It should be possible to make a complete audit of the test.

None of this is possible if we don't follow the specification, but omit or add activities as we please.

There is nothing wrong with getting new ideas for additional test cases

to improve the test specification during the execution. In fact we neither can, nor should, avoid it. But new ideas must go through the right channels, not just be acted out on the fly. The right channel in this context is an incident management system. New ideas for tests should be treated as incidents (enhancement requests) for the test. This is another reason why it is important to have the configuration management system in place before the test execution starts.

It is quite possible that some of the test execution time has been reserved for performing experienced based testing, where we don't use prespecified test procedures. These techniques are discussed in Section 4.4.

2.4.3.5 Identifying Failures

For each test case we execute the actual result should be logged and compared to the expected result, defined as part of the test case. This can be done in various ways depending on the formality of the test. For fairly informal testing a tick mark, √, is sufficient to indicate when the actual result matched the expected result. For more formal testing, for example, for safety-critical software, the authorities require that the actual result is recorded explicitly. This could be in the form of screen dumps, included reports, or simply writing the actual result in the log. This type of logging may also serve as part of the proof that the test has actually been executed.

We need to be very careful when we compare the expected result with the actual result, in order not to miss failures (called false positives) or report correct behavior as failures (called false negatives).

If the actual outcome does not comply with the expected outcome we have a failure on our hands. Any failure must be reported in the incident management system. The reported incident will then follow the defined incident life cycle. Incident reporting and handling is discussed in Chapter 7.

It is worth spending sufficient time reporting the incident we get. Too little time spent on reporting an incident may result in wasted time during the analysis of the incident. In the worst case it may be impossible to reproduce the failure, if we are not specific enough in reporting the circumstances and the symptoms.

Don't forget that the failure may be a symptom of a defect in our work products, like the test environment, the test data, the prerequisites, the expected result, and/or the way the execution was carried out. Such failures should also be reported in order to gather information for process improvement.

2.4.3.6 Test Execution Logging

As we execute, manually or by the use of tools, we must log what is going on. We must record the precise identification of what we are testing and the test

environment and test procedures we use. We must also log the result of the checking, as discussed above. Last but not least we must log any significant event that has an effect on the testing.

The recording of this information serves a number of purposes. It is indispensable in a professional and well-performed test.

The test execution may be logged in many different ways, often supported by a test management tool. Sometimes the event registration is kept apart in a test journal or diary.

The IEEE 829 standard suggests the following contents of a test log:

Test log identifier

1. Description of the test
2. Activity and event entries

It is handy and efficient if the test specification has built-in logging facilities that allow us to use it for test recording as we follow it for test execution. An example of this is shown here.

Ex.

Test Procedure: 3.6 (17)			
Purpose: This test suite tests ...			
Rationale: User requirement 82			
Prerequisites: The form ...			
Expected duration: 15 min.			
Execution time: ***Log when*** **Initials:** ***Log who***			
System: ***Identify object etc.*** **Result:** ***Log overall result***			
Case	**Input**	**Expected output**	**Actual output**
1.	Enter...		***Log result***

The information about which test procedures have been executed and with what overall result must be available at any given time. This information is used to monitor the progress of the testing.

The identification of the test object and the test specification may be used to ensure that possible confirmation testing after defect correction is done on the correct version of the test object (the new version) using the correct version of the test specification (the old or a new as the case might be).

The rationale—the tracing to the coverage items—can be used to calculate test coverage measures. These are used in the subsequent checking for test completion.

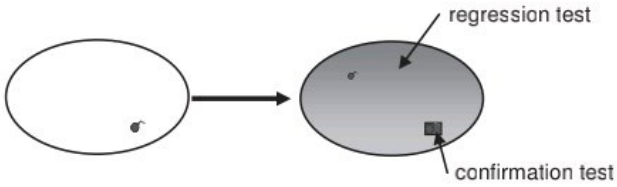
Information about who executed the test may be useful in connection with defect finding, for example, if it turns out to be difficult for the developer to reproduce or understand the reported failure.

2.4.3.7 Confirmation Testing and Regression Testing

During testing we get failures. In most cases the underlying defects are corrected and the corrected test object is handed over to the testers for confirmation. This is the situation where we iterate in the test process and go back to the test execution process. We go back to perform confirmation testing and regression testing.

Confirmation testing and regression testing are important activities in test execution. They can appear in all the test levels from component testing to (one hopes rarely) acceptance testing and even during maintenance of a product in operation.

These two types of change-related testing have one thing in common: they are executed after defect correction. Apart from that, they have very different goals.



In the figure above the test object with a defect is shown to the left. The defect has been unveiled in the testing. The defect has subsequently been corrected and we have got the new test object back again for testing; this is the one to the right.

What we must do now are confirmation testing and regression testing of the corrected test object.

Confirmation Testing

Confirmation testing is the first to be performed after defect correction. It is done to ensure that the defect has indeed been successfully removed. The test that originally unveiled the defect by causing a failure is executed again and this time it should pass without problems. This is illustrated by the dark rectangle in the place where the defect was.

Regression Testing

Regression testing may—and should—then be performed.

Regression testing is repetition of tests that have already been performed without problems to ensure that defects have not been introduced or uncovered as a result of the change. In other words it is to ensure the object under test has not regressed.

Courtesy of Grove Consultants.

This example shows a case of regression: A correction of a fault in a document using the "replace all" of the word "Author" with the word "Speaker" had an unintended effect in one of the paragraphs:

"... If you are providing the Presentation as part of your duties with your company or another company, please let me know and have a Speakerized representative of the company also sign this Agreement."

The amount of regression testing can vary from a complete rerun of all the test procedures that have already passed, to, well, in reality, no regression testing at all. The amount depends on issues such as:

- The risk involved
- The architecture of the system or product
- The nature of the defect that has been corrected

The amount of regression testing we choose to do must be justified in accordance with the strategy for the test.

Regression testing should be performed whenever something in or around the object under testing has changed. Fault correction is an obvious reason. There could also be others, more external or environmental changes, which could cause us to consider regression testing.

An example of an environment change could be the installation of a new version of the underlying database administration system or operating system. Experience shows that such updates may have the strangest effects on systems or products previously running without problems.

2.4.4 Metrics for Implementation and Execution

Metrics to be defined for the implementation and execution of the test implementation and execution activities may include:

- Number of created test environments over time
- Number of created test data over time
- Number of created test procedures over time
- Number of initiated test procedures over time
- Number of passed test procedures over time
- Number of failed test procedures over time
- Number of passed confirmation tests over time
- Number of test procedures run for regression testing over time
- Time spent on the various tasks

This will, of course, have to be compared to the estimates and schedule of the test implementation and execution activities.

2.5 Evaluating Exit Criteria and Reporting

Test execution, recording, control, retesting, and regression testing must be

continued until we believe that the exit criteria have been achieved. All the way we need to follow what is going on.

The purpose of the test progress and completion reporting is to stay in control of the testing and deliver the results of the testing activities in such ways that they are understandable and useful for the stakeholders.

The inputs on which this process is based are:

- Test plan
- Measurements from the test development and execution processes

The activities are:

- Comparing actual measurements with estimates and planned values
- Reporting test results

The output consists of:

- Presentation of test progress
- Test report

This is the second place from which iterations may occur

2.5.1 Input to Test Progress and Completion Reporting

The input from the level test plan that we need for this process is:

- Scheduling and staffing for the activities
- Exit criteria

2.5.2 Documentation of Test Progress and Completion Reporting

The documentation of the progress must be presented in various ways according to who is receiving it. The audience may be the customer, higher management, project management and participants, and testers.

Section 3.4.2 discusses presentation of monitoring information in great detail.

At the completion of each test level a test summary report should be produced. The ultimate documentation of completion is the final test summary report for the entire test assignment. The contents of a test summary report are described in Section 3.2.3.5.

2.5.3 Activities in Test Progress and Completion Reporting

The activities related to the test progress and completion reporting are discussed in the sections referenced above.

2.5.3.1 Checking for Completion

A check against the test exit criteria is mandatory before we can say that the testing is completed at any level. To warrant a stop it is important to ensure that the product has the required quality.

The exit criteria are tightly connected to the coverage items for the test, the test case design techniques used, and the risk of the product. The exit criteria therefore vary from test level to test level.

Examples of exit criteria are:

- Specified coverage has been achieved
- Specified number of failures found per test effort has been achieved
- No known serious faults
- The benefits of the system as it is are bigger than known problems

If the exit criteria are not met the test cannot just be stopped. An iteration in the test process must take place: We have to go back to where something can be repeated to ensure that the exit criteria are fulfilled.

In most cases additional test procedures are required. This means that the test analysis and design process must be revisited and more test cases and procedures added to increase coverage. These test procedures must then be executed, and the results recorded and checked. Finally the checking of the exit criteria must be completed.

Alternatively, the test plan may be revised to permit the relaxation (or strengthening) of test exit criteria.

Any changes to the test completion criteria must be documented, ideally having first identified the associated risk and agreed to the changes with the customer. Changing the test plan by adjusting the completion criteria should be regarded as an emergency situation and be very well accounted for.

When all test completion criteria are met and the report approved, the test object can be released. Release has different meanings at different points in the development life cycle:

- When the test is a static test the test object (usually a document) can be released to be used as the basis for further work.
- When the test is a test level for dynamic test the test object is progressively released from one test level to the next.
- Ultimately the product can be released to the customer.

2.5.4 Metrics for Progress and Completion Reporting

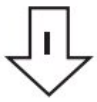
Metrics to be defined for the progress and control activities themselves may include:

- Number of tasks commenced over time
- Task completion percentage over time
- Number of task completed over time
- Time spent on each task over time

This will of course have to be compared to the estimates and schedule of the test progress and completion activities.

2.6 Test Closure

The purpose of the test closure activities is to consolidate experience and place test ware under proper control for future use.



The inputs on which this process is based are:

- Level test plan
- Test ware, including test environment

The overall procedure consists of the activities:

- Final check of deliveries and incident reports
- Secure storage/handover of test ware
- Retrospection

The output generated in this process is:

- Test experience report
- Configuration management documentation

2.6.1 Input to Test Closure

The input from the test plan that we need for this process is:

- Scheduling and staffing for the activities
- Planned deliveries

Furthermore we need all the test ware, both the test plans and specification, we have produced prior to test execution, the test environment, and the logs, incidents, and other reports we have produced during and after test execution. We also need the experiences made by all the participants and other stakeholders. These are often in the form of feelings and opinions of what has been going on.

2.6.2 Documentation of Test Closure

The documentation from this process is an experience report or a retrospective report from the retrospective meeting.

Other documentation will exist in the form it is specified in the organization's and/or customer's configuration management system.

2.6.3 Activities in Test Closure

2.6.3.1 Check Completion Again

Before we definitively close the door to the testing assignment we need to make extra sure that we have met the part of the exit criteria. This is both in terms of test coverage and deliveries we are to produce. If this is not in order or any discrepancies not clearly documented we'll have to make sure it is before we proceed.

2.6.3.2 Delivering and Archiving Test Ware

The test ware we have produced are valuable assets for the organization and should be handled carefully. For the sake of easy and economically sound future testing in connection with defect correction and development of new versions of the product we should keep the assets we have produced.

It is a waste of time and money not to keep the test ware we have produced.

If the organization has a well-working configuration management system this is what we must use to safeguard the test ware.

If such a system does not exist, we must arrange with those who are taking over responsibility for the product how the test ware must be secured. Those taking over could, for example, be a maintenance group or the customer.

2.6.3.3 Retrospective Meeting

The last thing we have to do is to report the experiences we have gained during our testing. The measurements we have collected should be analyzed and any other experiences collected and synthesized as well. This must be done in accordance with the approach to process improvement expressed in the test policy and the test strategy, as discussed in Section 3.2.

This is also a very valuable activity since the results of the testing can be the main indicators of where processes need to be improved. This can be all processes, from development processes (typically requirements development) over support processes (typically configuration management, not least for requirements) to the test process itself.

It is important that we as testers finish our testing assignment properly by producing an experience report.

For the sake of the entire process improvement activity, and hence the entire organization, it is important that higher management is involved and asks for and actively uses the test experience reports. Otherwise, the retrospective meetings might not be held, because people quickly get engrossed in new (test) projects and forget about the previous one.

2.6.4 Metrics for Test Closure Activities

Metrics to be defined for these activities may include number of tasks commenced over time, task completion percentage over time, number of tasks completed over time, and time spent on each task over time as for the other processes.

This will of course have to be compared to the estimates and schedule of the test closure activities.

Questions

1. Which three elements must always be defined for a process?
2. How do processes depend on each other?
3. What are the five activities (subprocesses) in the generic test process?
4. To which test levels and other test types does the generic test process apply?
5. Which iterations are embedded in the generic test process?
6. From where can we get inspiration for test process definitions?
7. What it the input to the test planning process?
8. What is the table of contents for a test plan suggested by IEEE 829?
9. Why is early planning a good idea?
10. What can the test basis be for each of the dynamic test levels?
11. What should be covered in the test approach description?
12. What is completion criteria?
13. What are the typical test deliveries?
14. What is the structure of a test specification according to IEEE 829?
15. What are the parameters we use to plan the test?
16. What is a work breakdown structure?
17. What are the testing roles we need to handle all test activities?
18. What are the activities in the test analysis and design process?
19. What should be in a test design according to IEEE 829?
20. What test design would be relevant for a system test?
21. What is a test condition?
22. How are test cases created?
23. What must be defined for each test case according to IEEE 829?
24. What is the expected result in a test case?
25. What could be used if the expected result cannot be determined easily?
26. What requirements types should we expect to find in a requirements specification?
27. What are the recommendations for expressing requirements as statements?
28. What is traceability?
29. What are the activities in the test implementation and execution process?
30. What is a test procedure?