CHAPTER

1

Basic Aspects of Software Testing

Contents

Testing is not an isolated activity, nor is it a development activity. Testing is a support activity: meaningless without the development processes and not producing anything in its own right: nothing developed entails nothing to test.

Testing is, however, a very important part of the life cycle of any product from the initial idea, during development, and in deployment until the product is taken out of deployment and disposed of.

Testing has it place intertwined with all these activities. Testing must find its place and fill it as well as possible.

1.1 Testing in the Software Life Cycle

The intention of product development is to somehow go from the vision of a product to the final product.

Copyright © 2008. Artech House. All rights reserved.

Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Book: sample.indb 1 2/19/08 8:12:28 PM

To do this a development project is usually established and carried out. The time from the initial idea for a product untill it is delivered is the development life cycle.

When the product is delivered, its real life begins. The product is in use or deployed until it is disposed of. The time from the initial idea for a product until it is disposed of is called the product life cycle, or software life cycle, if we focus on software products.

Testing is a necessary process in the development project, and testing is also necessary during deployment, both as an ongoing monitoring of how the product is behaving and in the case of maintenance (defect correction and possibly evolution of the product).

Testing fits into any development model and interfaces with all the other development processes, such as requirements definition and coding. Testing also interfaces with the processes we call supporting processes, such as, for example, project management.

Testing in a development life cycle is broken down into a number of test levels—for example component testing and system testing. Each test level has it own characteristics.

1.1.1 Development Models

Everything we do in life seems to follow a few common steps, namely: conceive, design, implement, and test (and possibly subsequent correction and retest).

The same activities are recognized in software development, though there are normally called:

- Requirements engineering;
- Design;
- Coding;
- Testing (possibly with retesting, and regression testing).

Building blocks

In software development we call the building blocks "stages," "steps," "phases," "levels," or "processes."

The way the development processes are structured is the development life cycle or the development model. A life cycle model is a specification of the order of the processes and the transition criteria for progressing from one process to the next, that is, completion criteria for the current process and entry criteria for the next.

Software development models provide guidance on the order in which the major processes in a project should be carried out, and define the conditions for progressing to the next process. Many software projects have experienced problems because they pursued their development without proper regard for the process and transition criteria.

Copyright © 2008. Artech House. All rights reserved.

Book1.sample.indb 2 2/19/08 8:12:28 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

The reason for using a software development model is to produce better quality software faster. That goal is equal for all models. *Using any model is better than not using a model.*

A number of software development models have been deployed throughout the industry over the years. They are usually grouped according to one of the following concepts:

- Sequential;
- Iterative;
- Incremental.

The building blocks—the processes—are the same; it is only a matter of their length and the frequency with which they are repeated.

The sequential model is characterized by including no repetition other than perhaps feedback to the preceding phase. This makeup is used in order to avoid expensive rework.

1.1.1.1 Sequential Models

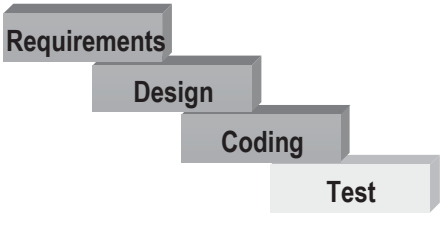
The *assumptions* for sequential models are:

- The customer knows what he or she wants.
- The requirements are frozen (changes are exceptions).
- Phase reviews are used as control and feedback points.

The characteristics of a successful sequential development project are:

- Stable requirements;
- Stable environments;
- Focus on the big picture;
- One, monolithic delivery.

Historically the first type of sequential model was *the waterfall model.* A pure waterfall model consists of the building blocks ordered in one sequence with testing in the end.

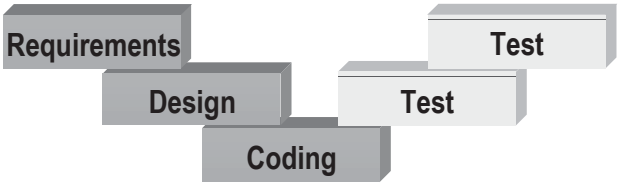


Copyright © 2008. Artech House. All rights reserved.

Book1.samp.indb 3
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.
2/19/08 8:12:29 PM

The goals of the waterfall model are achieved by enforcing fully elaborated documents as phase completion criteria and formal approval of these (signatures) as entry criteria for the next.

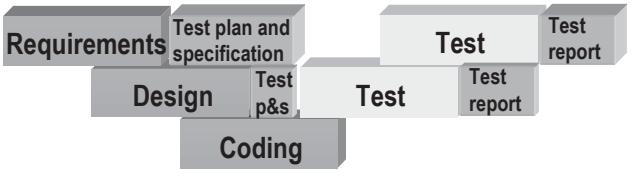
The V-model is an expansion of the pure waterfall model introducing more test levels, and the concept of testing not only being performed at the end of the development life cycle, even though it looks like it.



The V-model describes a course where the left side of the V reflects the processes to be performed in order to produce the pieces that make up the physical product, for example, the software code. The processes on the right side of the V are test levels to ensure that we get what we have specified as the product is assembled.

The pure V-model may lead you to believe that you develop first (the left side) and then test (the right side), but that is not how it is supposed to work.

A *W-model* has been developed to show that the test work, that is, the production of testing work products, starts as soon as the basis for the testing has been produced. Testing includes early planning and specification and test execution when the objects to test are ready. The idea in the V-model and the W-model is the same; they are just drawn differently.



When working like this, we describe what the product must do and how (in the requirements and the design), and at the same time we describe how we are going to test it (the test plan and the specification). This means that we are starting our testing at the earliest possible time.

The planning and specification of the test against the requirements should, for example, start as soon as the requirements have reached a reasonable state.

Copyright © 2008. Artech House. All rights reserved.

Book.samfet.indb 4
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:12:29 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

A W-model-like development model provides a number of advantages:

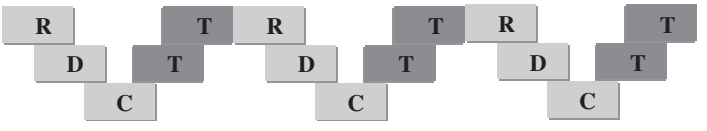
- More time to plan and specify the test
- Extra test-related review of documents and code
- More time to set up the test environment(s)
- Better chance of being ready for test execution as soon as something is ready to test

For some classes of software (e.g., safety critical systems, or fixed-price contracts), a W-model is the most appropriate.

1.1.1.2 Iterative and Incremental Models

In iterative and incremental models the strategy is that frequent changes should and will happen during development. To cater for this the basic processes are repeated in shorter circles, iterations. These models can be seen as a number of mini W-models; testing is and must be incorporated in every iteration within the development life cycle.

This is how we could illustrate an iterative or incremental development model.



The goals of an iterative model are achieved through various prototypes or subproducts. These are developed and validated in the iterations. At the end of each iteration an operational (sub)product is produced, and hence the product is expanding in each iteration. The direction of the evolution of the product is determined by the experiences with each (sub)product.

Note that the difference between the two model types discussed here is:

- In *iterative development* the product is not released to the customer until all the planned iterations have been completed.
- In *incremental development* a (sub)product is released to the customer after each iteration.

The *assumptions* for an iterative and incremental model are:

- The customer cannot express exactly what he or she wants.
- The requirements will change.
- Reviews are done continuously for control and feedback.

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 5
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.
2/19/08 8:12:30 PM

The characteristics of a successful project following such a model are:

- Fast and continuous customer feedback;
- Floating targets for the product;
- Focus on the most important features;
- Frequent releases.

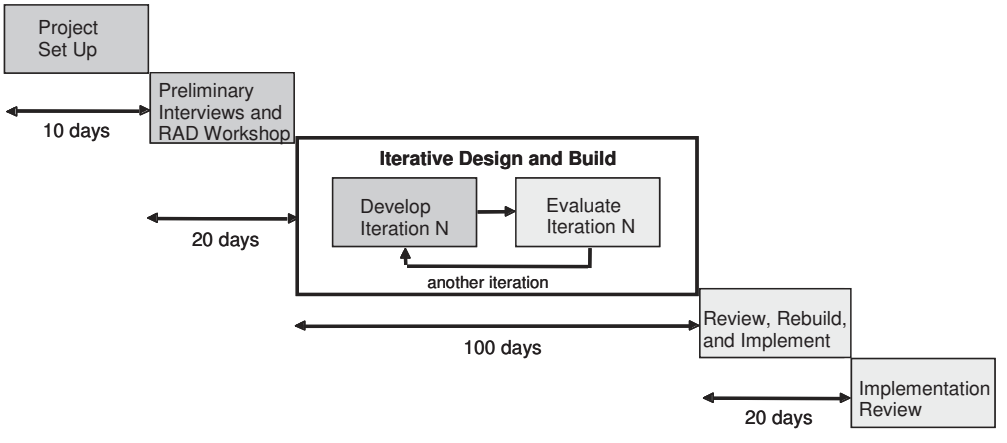
The iterative/incremental model matches situations in which the customers say: "I can't tell you what I want, but I'll know it when I see it"—the last part of the sentence often expressed as "IKIWISI."

These models are suited for a class of applications where there is a close and direct contact with the end user, and where requirements can only be established through actual operational experience.

A number of more specific iterative models are defined. Among these the most commonly used are the RAD model and the Spiral model.

The *RAD model* (Rapid Application Development) is named so because it is driven by the need for rapid reactions to changes in the market. James Martin, consultant and author, called the "guru of the information age", was the first to define this model. Since then the term RAD has more or less become a generic term for many different types of iterative models.

The original RAD model is based on development in timeboxes in few—usually three—iterations on the basis of fundamental understanding of the goal achieved before the iterations start. Each iteration basically follows a waterfall model.



Copyright © 2008. Artech House. All rights reserved.

Book1.sample.indb 6 2/19/08 8:12:31 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

When the last iteration is finished, the product is finalized and implemented as a proper working product to be delivered to the customer.

Barry Boehm, TRW Professor of Software Engineering at University of Southern California, has defined a so-called *Spiral Model*. This model aims at accommodating both the waterfall and the iterative model. The model consists of a set of full cycles of development, which successively refines the knowledge about the future product. Each cycle is risk driven and uses prototypes and simulations to evaluate alternatives and resolve risks while producing work products. Each cycle concludes with reviews and approvals of fully elaborated documents before the next cycle is initiated.

The last cycle, when all risks have been uncovered and the requirements, product design, and detailed design approved, consists of a conventional waterfall development of the product.

In recent years a number of incremental models, called *evolutionary* or *agile development models,* have appeared. In these models the emphasis is placed on values and principles, as described in the "Manifesto of Software Development." These are:

- Individuals and interactions are valued over processes and tools
- Working software is valued over comprehensive documentation
- Customer collaboration is valued over contract negotiation
- Responding to change is valued over following a plan

One popular example of these models is the eXtreme Programming model, (XP). In XP one of the principles is that the tests for the product are developed first; the development is test-driven.

The development is carried out in a loosely structured small-team style. The objective is to get small teams (3–8 persons) to work together to build products quickly while still allowing individual programmers and teams freedom to evolve their designs and operate nearly autonomously.

These small teams evolve features and whole products incrementally while introducing new concepts and technologies along the way. However, because developers are free to innovate as they go along, they must synchronize frequently so product components all work together.

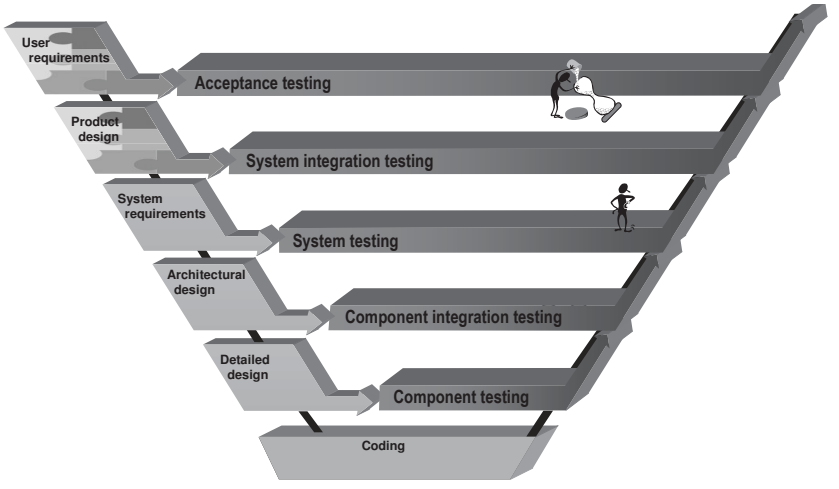
Testing is perhaps even more important in iterative and incremental development than in sequential development. The product is constantly evolved and extensive regression testing of what has previously been agreed and accepted is imperative in every iteration.

Copyright © 2008. Artech House. All rights reserved.

Book-sample.indb 7 2/19/08 8:12:31 PM
Mette Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

1.1.2 Dynamic Test Levels

In the V-model, and hence in basically all development, each development process has a corresponding dynamic test level as shown here.



The V-model used here includes the following dynamic test levels:

- Acceptance testing—based on and testing the fulfillment of the user requirements;
- System testing—based on and testing the fulfillment of the (software) system requirements;
- Component integration testing—based on and testing the implementation of the architectural design;
- Component testing—based on and testing the implementation of the detailed design.

Note that coding does not have a corresponding test level; it is not a specification phase, where expectations are expressed, but actual manufacturing! The code becomes the test object in the dynamic test levels.

There is no standard V-model. The V-model is a principle, not a fixed model. Each organization will have to define its own so that it fits with the nature of the products and the organization. The models can have different make-ups, that is there may be more or less specification phases on the left side and hence testing levels on the right side, and/or the phases and levels may be named differently in different organizations.

Copyright © 2008. Artech House. All rights reserved.

Book-samtest.indb 8 2/19/08 8:12:32 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

In cases where the final product is part of or in itself a complex product it is necessary to consider more integration test levels.

In the case of a system of systems development project, described in Section 1.2.1, we will need a system integration test level.

Sometimes the product we are developing consists of a number of different types of systems, like for example:

- Software
- Hardware
- Network
- Data
- Services

In such cases there will be product design specification phases to distribute the requirements on the different systems in the beginning of the development life cycle and we will therefore need more or more integration test levels, such as, for example, hardware-software system integration and software-data system integration.

Note: the puzzle does NOT indicate possible interfaces between systems, only the fact that a product may be made up of different types of systems.

We could also be producing a product that is going to interface with system(s) the customer already has running. This will require a customer product integration test level.

No matter how many test levels we have, each test level is different from the others, especially in terms of goals and scope.

The organizational management must provide test strategies specific to each of the levels for the project types in the organization in which the testing is anchored. The contents of a test level strategy are discussed in Section 3.2.2.

Based on this the test responsible must produce test plans specific for each test level for a specific project. The contents of a test plan are discussed in Section 3.2.3.

The specific test plans for the test levels for a specific project should outline the differences between the test levels based on the goals and scope for each.

The fundamental test process is applicable for all the test levels. The test process is described in detail in Chapter 2.

The dynamic test levels in the V-model used here are discussed next.

1.1.2.1 Component Testing

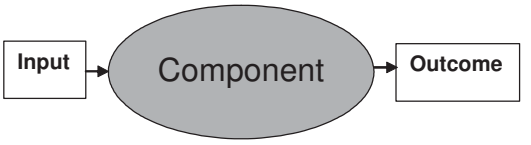
Component testing is the last test level where work, that is, planning, can start, the first where test execution can start, and therefore also the first to be finished.

Copyright © 2008. Artech House. All rights reserved.

Book-sample.indb 9
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:12:33 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

The *goal* is to find defects in the implementation of each component according to the detailed design of the component.

The test object is hence individual components in isolation, and the basis documentation is the detailed design, and sometimes also other documentation like the requirements specification.



It is not always easy to agree on what a component is. A component could be what is contained in a compilable file, a subroutine, a class, or … the possibilities are legion. The important thing in an organization is to define "a component"—it is less important what a component is defined as.

The scope for one component test is the individual component and the full scope of the component testing could be all components specified in the design, though sometimes only the most critical components may be selected for component testing.

An overall *component test plan* should be produced specifying the order in which the testing of the components is to take place. If this is done sufficiently early in the development, we as testers may be able to influence the development order to get the most critical components ready to test first. We also need to consider the subsequent component integration testing, and plan for components with critical interfaces to be tested first. For each component a very short plan (who, when, where, and completion criteria) and a test specification should be produced.

The assignment of the responsibility for the component testing depends on the level of independence we need. The lowest level of independence is where the manufacturer—here the developer—tests his or her own product. This often happens in component testing. The next level of independence is where a colleague tests his or her colleague's product. This is advisable for component testing. The level of independence to use is guided by the risk related to the product. Risk management is discussed in Section 3.5.

The *techniques* to use in component testing are functional (black-box) techniques and structural (white-box) techniques. Most often tests are first designed using functional techniques. The coverage is measured and more tests can be designed using structural techniques if the coverage is not sufficient to meet the completion criteria.

The code must never be used as the basis documentation from which to derive the expected results.

Nonfunctional requirements or characteristics, such as memory usage,

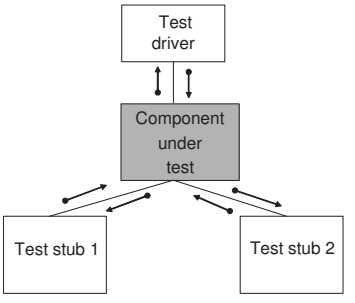
Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 10 2/19/08 8:12:33 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

defect handling, and maintainability may also be tested at the component testing level.

To isolate a component, it is necessary to have a driver to be able to execute the component. It is also usually necessary to have a stub or a simulator to mimic or simulate other components that interface with the component under test. Test stubs are sometimes referred to as test harness.

The needs for test drivers and stubs must be specified as part of the specification of the test environment. Any needed driver and stubs must, of course, be ready before each individual component testing can start.



Many tools support component testing. Some are language-specific and can act as drivers and stubs to facilitate the isolation of the component under test. Tools are discussed in Chapter 9.

Component *test execution* should start when the component has been deemed ready by the developer, that is when it fulfilles the entry criteria. The least we require before the test execution can start is that the component can compile. It could be a very good idea to require that a static test and/or static analysis has been performed and approved on the code as entry criteria for the component test execution.

Measures of time spent on the testing activities, on defects found and corrected, and on obtained coverage should be collected. This is sometimes difficult because component testing is often performed as an integrated development/testing/debugging activity with no registration of defects and very little if any reporting. This is a shame because it deprives the organization of valuable information about which kinds of defects are found and hence the possibility for introducing relevant process improvement.

The component testing for each individual component must stop when the completion criteria specified in the plan have been met. For each component a very short summary report should be produced.

A summary *report* for the collection of components being tested should be produced when the testing has been completed for the last component.

Any *test procedures should be kept*, because they can be very useful for later confirmation testing and regression testing. Drivers and stubs should be kept for the same reason, and because they can be useful during integration testing as well.

The goals of integration testing are to find defects in the interfaces and invariants between interacting entities that interact in a system or a product. Invariants are substates that should be unchanged by the interaction between two entities.

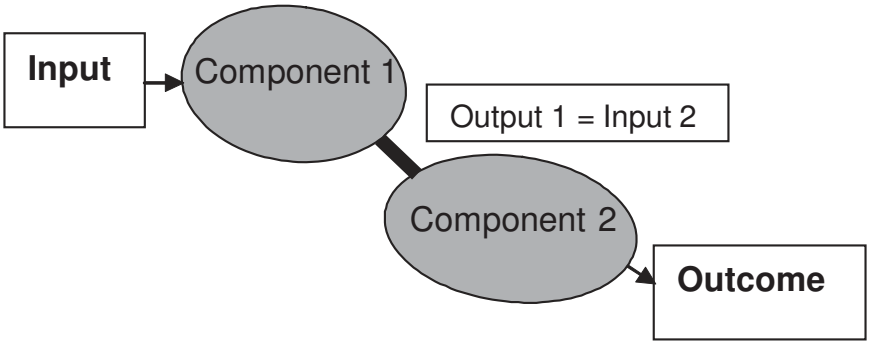
Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 11 2/19/08 8:12:34 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

1.1.2.2 Integration Testing

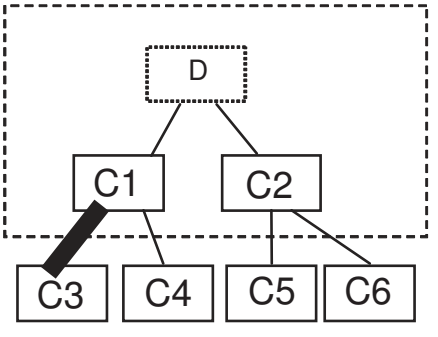
The objective is not to find defects inside the entities being integrated—the assumption being that these have already been found during previous testing.

The entities to integrate may be components as defined in the architectural design or different systems as defined in the product design. The principles for integration testing are the same no matter what we are integrating.



For the collection of interfaces to test an overall integration test plan should be produced specifying, among other things, the order in which this testing is to take place. There are four different strategies for the testing order in integration testing:

- Top down;
- Bottom up;
- Functional integration;
- Big-bang.



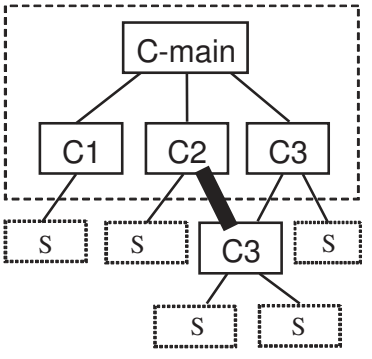
In top-down integration the interfaces in the top layer in the design hierarchy are tested first, followed by each layer going downwards. The main program serves as the driver.

This way we quickly get a "shell" created. The drawback is that we (often) need a large number of stubs.

Copyright © 2008. Artech House. All rights reserved.

Book1.samp1.indb 12 2/19/08 8:12:34 PM
Mette Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

In bottom-up integration the interfaces in the lowest level are tested first. Here higher components are replaced with drivers, so we may need many drivers. This integration strategy enables early integration with hardware, where this is relevant.



In functional integration we integrate by functionality area; this is a sort of vertically divided top-down strategy. We quickly get the possibility of having functional areas available.

In big-bang integration we integrate most or everything in one go. At first glance it seems like this strategy reduces the test effort, but it does not—on the contrary. It is impossible to get proper coverage when testing the interfaces in a big-bang integration, and it is very difficult to find any defects in the interfaces, like looking for a needle in a haystack. Both top-down and bottom-up integration often end up as big-bang, even if this was not the initial intention.

For each interface a *very short plan* (who, when, where, and completion criteria) and a test specification should be produced. Often one of the producers of the entities to integrate has the responsibility for that integration testing, though both should be present.

Both the formality and the level of independence is higher for system integration testing than for component integration, but these issues should not be ignored for component integration testing.

The *techniques* to use must primarily be selected among the structural techniques, depending on the completion criteria defined in the plan. Nonfunctional requirements or characteristics, such as performance, may also be tested at the integration testing level.

The necessary drivers or stubs must be specified as part of the environment and developed before the integration testing can start. Often stubs from a previous test level, for example, component testing, can be reused.

The *execution* of the integration testing follows the completion of the testing of the entities to integrate. As soon as two interacting entities are tested, their integration test can be executed. There is no need to wait for all entities to be tested individually before the integration test execution can begin.

Measures of time spent on the testing, on defects found and corrected, and on coverage should be collected.

The integration testing for each individual interface must stop when the completion criteria specified in the plan have been met. A very short test report should be produced for each interface being tested. We must keep on integrating and testing the interfaces and the invariants until all the entities

Copyright © 2008. Artech House. All rights reserved.

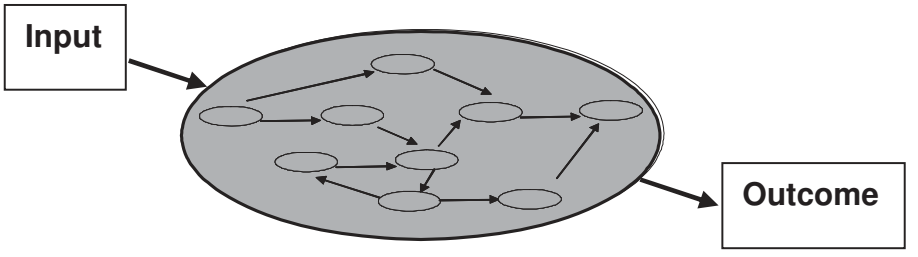
Book: samfer.indb 13
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:12:35 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

have been integrated and the overall completion criteria defined in the integration test plan have been met.

A summary report for the collection of interfaces being tested should be produced when the testing has been completed for the last interface.

1.1.2.3 System Testing

The goal of system testing is to find defects in features of the system compared to the way it has been defined in the software system requirements. The test object is the fully integrated system.



The better the component testing and the component integration testing has been performed prior to the system testing, the more effective is the system testing. All too often system testing is impeded by poor or missing component and component integration testing.

A comprehensive system test plan and system test specification must be produced.

The system test specification is based on the system requirements specification. This is where all the expectations, both the functional and the non-functional should be expressed. The functional requirements express what the system shall be able to do—the functionality of the system. The non-functional requirements express how the functionality presents itself and behaves. In principle the system testing of the two types of requirements is identical. We test to get information about the fulfillment of the requirements.

The techniques to use will most often be selected among the functional techniques, possibly supplemented with experience-based techniques (exploratory testing, for example), depending on the completion criteria defined in the plan. *Experience-based test techniques should never be the only techniques used in the system testing.*

The *execution* of system test follows the completion of the entire component integration testing. It is a good idea to also require that a static test

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 14
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:12:35 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

has been performed on the requirements specification and on the system test specification before execution starts.

Many tools support system testing. Capture/replay tools and test management tools are especially useful to support the system testing.

Measures of time spent on the testing, on faults found and corrected, and on coverage should be collected. The system testing must stop when the completion criteria specified in the plan have been met.

A system test report should be produced when the system testing has been completed.

1.1.2.4 Acceptance Testing

The acceptance testing is the queen's inspection of the guard. The goal of this test level is not, like for all the other ones, to find defects by getting the product to fail. At the acceptance test level the product is expected to be working and it is presented for acceptance.

The customer and/or end users must be involved in the acceptance testing. In some cases they have the full responsibility for this testing; in other cases they just witness the performance.

In the acceptance testing the test object is the entire product. That could include:

- Business processes in connection with the new system;
- Manual operations;
- Forms, reports, and so forth;
- Document flow;
- Use cases and/or scenarios.

The techniques are usually mostly experience-based, where the future users apply their domain knowledge and (hopefully) testing skills to the validation of the product. Extracts of the system test specification are sometimes used as part of the acceptance test specification.

An extra benefit of having representatives of the users involved in the acceptance testing is that it gives these users a detailed understanding of the new system—it can help create ambassadors for the product when it is brought into production.

There may be a number of acceptance test types, namely:

- Contract acceptance test;
- Alpha test;
- Beta test.

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 15
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:12:36 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

The contract acceptance test may also be called factory acceptance test. This test must be completed before the product may leave the supplier; the product has to be accepted by the customer. It requires that clear acceptance criteria have been defined in the contract. A thorough registration of the results is necessary as evidence of what the customer acceptance is based on.

An alpha test is usage of the product by representative users at the development site, but reflecting what the real usage will be like. Developers must not be present, but extended support must be provided. The alpha test is not used particularly often since it can be very expensive to establish a "real" environment. The benefits rarely match the cost.

A beta test is usage of the product by selected (or voluntary) customers at the customer site. The product is used as it will be in production. The actual conditions determine the contents of the test. Also here extended support of the users is necessary. Beta tests preferably run over a longer period of time. Beta tests are much used for off-the-shelf products—the customers get the product early (and possibly cheaper) in return for accepting a certain amount of immaturity and the responsibility for reporting all incidents.

1.1.3 Supporting Processes

No matter how the development model is structured there will always be a number of supporting activities, or supporting processes, for the development.

The primary supporting processes are:

- Quality assurance;
- Project management;
- Configuration management.

These processes are performed during the entire course of the development and support the development from idea to product.

Other supporting processes may be:

- Technical writing (i.e., production of technical documentation);
- Technical support (i.e., support of environment including tools).

The supporting processes all interface with the test process.

Testing is a product quality assurance activity and hence actually part of the supporting processes. This is in line with the fact that testing is meaningless without the development processes and not producing anything in its own right: nothing developed entails nothing to test.

The test material, however, is itself subject to quality assurance or testing, so testing is recursive and interfaces with itself. Testing also interfaces with

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 16 2/19/08 8:12:36 PM
Mette Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

project management and configuration management as discussed in detail in the following.

Testing also interfaces with technical writing. The documentation being written is an integrated part of the final product to be delivered and must therefore also be subject to quality assurance (i.e., to static testing).

When the product—or an increment—is deployed, it transfers to the maintenance phase. In this phase corrections and possibly new features will be delivered at defined intervals, and testing plays an important part here.

1.1.3.1 Product Quality Assurance

It is not possible to test quality into a product when the development is close to being finished. The quality assurance activities must start early and become an integrated part of the entire development project and the mindset of all stakeholders.

Quality assurance comprises four activities:

- Definition of quality criteria
- Validation
- Verification
- Quality reporting

Note that the validation is not necessarily performed before the verification; in many organizations it is the other way around, or in parallel.

First of all, the *Quality criteria* must be defined. These criteria are the expression of the quality level that must be reached or an expression of "what is sufficiently good." These criteria can be very different from product to product. They depend on the business needs and the product type. Different quality criteria will be set for a product that will just be thrown away when it is not working than for a product that is expected to work for many years with a great risk of serious consequences if it does not work.

There are two quality assurance activities for checking if the quality criteria have been met by the object under testing, namely:

- Validation;
- Verification.

They have different goals and different techniques. The object to test is delivered for validation and verification from the applicable development process.

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 17
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebook Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:12:37 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

Validation is the assessment of the correctness of the product (the object) in relation to the users' needs and requirements.

You could also say that validation answers the question: *"Are we building the correct product?"*

Validation must determine if the customer's needs and requirements are correctly captured and correctly expressed and understood. We must also determine if what is delivered reflects these needs and requirements.

When the requirements have been agreed upon and approved, we must ensure that during the entire development life cycle:

- Nothing has been forgotten.
- Nothing has been added.

It is obvious that if something is forgotten, the correct product has not been delivered. Is does, however, happen all too often, that requirements are overlooked somewhere in the development process. This costs money, time, and credibility.

On the surface it is perhaps not so bad if something has been added. But it does cost money and affect the project plan, when a developer—probably in all goodwill—adds some functionality, which he or she imagines would be a benefit for the end user.

What is worse is that *the extra functionality will probably never be tested* in the system and acceptance test, simply because the testers don't know anything about its existence. This means that the product is sent out to the customers with some untested functionality and this will lie as a mine under the surface of the product. Maybe it will never be hit, or maybe it will be hit, and in that case the consequences are unforeseeable.

The possibility that the extra functionality will never be hit is, however, rather high, since the end user will probably not know about it anyway.

Validation during the development process is performed by analysis of trace information. If requirements are traced to design and code it is an easy task to find out if some requirements are not fulfilled, or if some design or code is not based on requirements.

The ultimate validation is the user acceptance test, where the users test that the original requirements are implemented and that the product fulfills its purpose.

Verification, the other quality assurance activity, is the assessment of whether the object fulfills the specified requirements.

Verification answers the question: *"Are we building the product correctly?"*

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 18
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:12:37 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

The difference between validation and verification can be illustrated like this:

Validation confirms that a required calculation of discount has been designed and coded in the product.
Verification confirms that the implemented algorithm calculates the discount as it is supposed to in all details.

A number of techniques exist for verification. The ones to choose depend on the test object.

In the early phases the test object is usually a document, for example in the form of:

- Plans;
- Requirements specification;
- Design;
- Test specifications;
- Code.

The verification techniques for these are the static test techniques discussed in Chapter 6:

- Inspection;
- Review (informal, peer, technical, management);
- Walk-through.

Once some code has been produced, we can use static analysis on the code as a verification technique. This is not executing the code, but verifying that it is written according to coding standards and that is does not have obvious data flow faults. Finally, dynamic testing where the test object is executable software can be used.

We can also use dynamic analysis, especially during component testing. This technique reveals faults that are otherwise very difficult to identify. Dynamic analysis is described in Section 4.6.

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 19
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:12:37 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

A little memory hint:

Validation

Correct

a comes before e
t comes before y

Verification

Correctly

Quality assurance reports on the findings and results should be produced.

If the test object is not found to live up to the quality criteria, the object is returned to development for correction. At the same time incident reports should be filled in and given to the right authority.

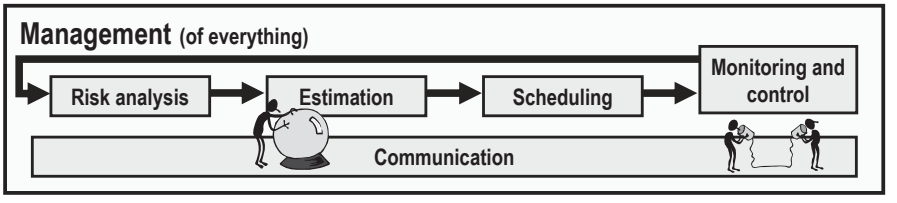
Once the test object has passed the validation and verification, it should be placed under configuration management.

1.1.3.2 Project Management

It is obviously important that the development process and the supporting processes are managed and controlled during the entire project. Project management is the supporting process that takes care of this, from the first idea to the release.

The most important activities in project management are:

- Risk analysis;
- Estimation;
- Scheduling;
- Monitoring and control;
- Communication.



Copyright © 2008. Artech House. All rights reserved.

Book-samler.indb 20 2/19/08 8:12:38 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Test management is subordinated to project management.

The estimation, risk analysis, and scheduling of the test activities will either have to be done in cooperation with the project management or by the test manager and consolidated with the overall project planning. The results of the monitoring and control of the test activities will also have to be coordinated with the project management activities.

The project management activities will not be discussed further here.

The corresponding test management activities are described in detail in Chapter 3.

1.1.3.3 Configuration Management

Configuration management is another supporting process with which testing interacts. The purpose of configuration management is to establish and maintain integrity of work products and product.

Configuration management can be defined as:

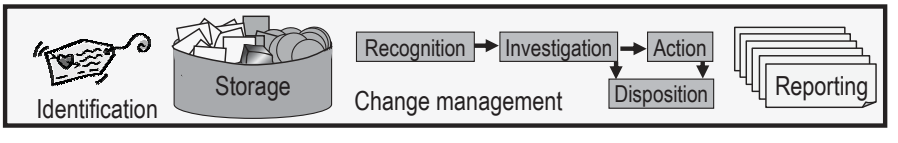
- Unique identification;
- Controlled storage;
- Change management (recognition, investigation, action, and disposition);
- Status reporting.

for selected

- Work products;
- Product components;
- Products.

during the entire life time of the product.

An object under configuration management is called a configuration item.



The purpose of *identification* is to uniquely identify each configuration item and to specify its relations to the outside world and to other configuration items. Identification is one of the cornerstones of configuration management, as it is impossible to control something for which you don't know the identity.

Each organization must define the conventions for unique identification of the configuration items.

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 21
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:12:48 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

Test cases
10.3.1.6 (80) Test for correct bank identity number 1.A
The identification encompasses:
Current section number in document: 10.3.1.6
Running unique number: 80
Version of test case: 1.A

The purpose of *storage* is to ensure that configuration items don't disappear or are damaged. It must also be possible to find the items at any time and have them delivered in the condition in which we expect to find them.

Storage is something physical. Items that are stored are physically present at a specific place. This place is often called the library, or the controlled library.

Configuration items are released from storage to be used as the basis for further work. *Usage is all imaginable deployment of configuration items without these being changed*, not just usage of the final product by the final users.

Usage may for instance be a review, if a document is placed under configuration management in the form of a draft and subsequently has to be reviewed.

It may be testing of larger or minor parts of the system, integration of a subcomponent into a larger component, or proper operation or sale of a finished product.

Configuration items released from storage must not be changed, ever! But new versions may be issued as the result of change control.

The purpose of *change management or change control* is to be fully in control of all change requests for a product and of all implemented changes. Any change should be traced to the configuration item where the change has been implemented.

The initiation of change control is the occurrence of an incident. Incident management is discussed in Chapter 7.

The purpose of *status reporting* is to make available the information necessary for effective management of the development, testing, and maintenance of a product, in a useful and readable way.

Configuration management can be a well of information.

A few words about the concept of a *configuration item* are needed here. In principle everything may be placed under configuration management. The following list shows what objects may become configuration items, with the emphasis on the test ware.

- Test material: Test specifications, test data(base), drivers, and stubs
- Environments: Operating systems, tools, compilers, and linkers

Copyright © 2008. Artech House. All rights reserved.

Book: samlet.indb 22 2/19/08 8:12:49 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

- Technical documentation: Requirements, design, and technical notes
- Code: Source code, header files, include files and system libraries
- Project documentation: User manuals, build scripts, data, event registrations, installation procedures, and plans
- Administrative documents: Letters, contracts, process description, sales material, templates, and standards
- Hardware: Cables, mainframe, PC, workstation, network, storage, and peripherals

1.1.3.4 Technical Writing

Technical writing is a support process much used in the United Kingdom. Other European countries do not use technical writers that much—here the developers, testers, and the rest of the project team are left to their own devices.

Technical writers are people with special writing skills and education. They assist other staff members when difficult issues need to be made clear to the intended audience in writing.

We as testers interface with technical writers in two ways:

- We subject their work to static tests.
- We use their work in our testing.

We can of course also use their skills as writers, but that does not happen very often. Testers usually write for other testers and for a technical audience.

1.2 Product Paradigms

The use of computers to assist people in performing tasks has developed dramatically since the first huge (in physical size) computers were invented around the middle of the last century. The first computers were about the size of a family home and you could only interact with them via punch cards or tape and printed output. Those were the days.

Today we as testers may have to cope with a number of different product types or product paradigms, and with different development paradigms and coding languages. Not all of us encounter all of them, but it is worth knowing a little bit about the challenges they each pose for us.

We always need to be aware of the product and development paradigm used for the (testing) projects we are involved in. We must tailor our test approach and detailed test processes to the circumstances and be prepared to tackle any specific obstacles caused by these as early as possible.

A few significant product paradigms are discussed here.

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 23
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:12:49 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

1.2.1 Systems of Systems

A system of systems is the concept of integrating existing systems into a single information system with only limited new development. The concept was first formulated by the American Admiral Owens in his book *Lifting the Fog of War*. The concept was primarily used for military systems but is spreading more and more to civilian systems as well.

The idea is to use modern network technologies to exploit the information spread out in a number of systems by combining and analyzing it and using the results in the same or other systems to make these even more powerful.

A tiny example of a system of systems is a sprinkling system at a golf course. The gardener can set the sprinkling rate for a week at the time. Using a network connection this system is linked to a system at the meteorological institute where hours of sunshine, average temperatures, and rainfall are collected. This information is sent to a small new system, which calculates the needed sprinkling rate on a daily basis and feeds this into the sprinkling system automatically. The gardener's time, water, the occasional flooding, and the occasional drying out of the green is saved.

Systems of systems are complicated in nature. The final system is usually large and complex as each of the individual systems may be. Each of the individual systems may in itself consist of a number of different subsystems, such as software, hardware, network, documentation, data, data repository systems, license agreements, services (e.g., courses and upgrades), and descriptions of manual processes. Few modern systems are pure software products, though they do exist.

Even if the individual systems are not developed from scratch these systems pose high demands on supporting the supporting processes, especially project management, but also configuration management and product quality assurance. In the cases where some or all of the individual systems are being developed as part of the construction of a system of systems this poses even higher demands in terms of communication and coordination.

From a testing point of view, there are at least three important aspects to take into account when working with systems of systems:

- System testing of the individual systems
- Integration testing of systems
- Regression testing of systems and integration

Copyright © 2008. Artech House. All rights reserved.

Book: samfer.indb 24
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:12:49 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

A system of systems is only as strong as the weakest link, and the completion criteria for the system testing of each individual system must reflect the quality expectations toward the complete system of systems. The system testing of each of the individual systems is either performed as part of the project, or assurance of its performance must be produced, for example in the form of test reports from the producer.

Systems of systems vary significantly in complexity and may be designed in hierarchies of different depths, ranging from a two-layer system where the final system of systems is composed of a number of systems of the same "rank" to many-layered (system of (systems of (systems of systems))). Integration of the systems must be planned and executed according to the overall architecture, observing the integration testing aspects discussed in Section 1.1.2.

It is inevitable that defects will be found during system and integration testing of systems of systems, and significant iterations of defect correction, confirmation testing, and not least regression testing must be anticipated and planned for. *Strict defect handling is necessary to keep this from getting out of control,* resulting, for example, in endless correction and recorrection circles.

Systems of systems may well contain systems of types where special care and considerations need to be made for testing. Examples may be:

- Safety-critical systems
- Large mainframe systems
- Client-server systems
- Web-based systems
- PC-based systems
- Web-based systems
- Embedded systems
- Real-time systems
- Object-oriented development

1.2.2 Safety-Critical Systems

Any system presents some risk to its owners, users, and environment. Some present more than others, and those that present the most risk are what we call safety-critical systems.

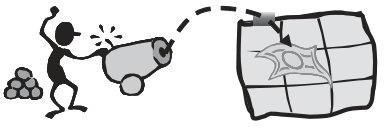
The risk is a threat to something valuable. All systems either have something of value, which may be jeopardized, inside them, or their usage may jeopardize some value outside them. A system should be built to protect the values both from the result of ordinary use of the system and from the result of malicious attacks of various kinds.

Copyright © 2008. Artech House. All rights reserved.

Book: samfer.indb 25
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:12:49 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

A typical categorization of values looks at values concerning:

- Safety
- Economy
- Security
- Environment



Many regulatory standards address how to determine the safety criticality of systems and provide guidelines for the corresponding testing. Some of them (but probably not all) are:

- CEI/IEC 61508—Functional safety of electrical/electronic/programmable safety-related systems
- DO-178-B—Software considerations in airborne systems and equipment certification
- pr EN 50128—Software for railway control and protection systems
- Def Stan 00-55—Requirements for safety-related software in defense equipment
- IEC 880—Software for computers in the safety systems of nuclear power stations
- MISRA—Development guidelines for vehicle-based software
- FDA—American Food and Drug Association (pharmaceutical standards)

The standards are application-specific, and that can make it difficult to determine what to do if we have to do with multidisciplinary products. Nonetheless, standards do provide useful guidance. The most generic of the standards listed above is IEC 61508; this may always be used if a system does not fit into any of the other types.

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 26 2/19/08 8:12:50 PM
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

All the standards operate with so-called software integrity levels (SILs).

This table shows an example of a classification.

Ex.

SIL / Value	A (100.000.000)	B (100.000)	C (100)	D (1)
Safety	Many people killed	Human lives in danger	Damage to physical objects; risk of personal injury	Insignificant damage to things; no risk to people
Economy	Financial catastrophe (the company must close)	Great financial loss (the company is threatened)	Significant financial loss (the company is affected)	Insignificant financial loss
Security	Destruction/disclosure of strategic data and services	Destruction/ disclosure of critical data and services	Faults in data	No risk for data
Environment	Comprehensive and irreparable damage to the environment	Reparable, but comprehensive damage to the environment	Local damage to the environment	No environmental risk

The concept of SILs allows a standard to define a hierarchy of levels of testing (and development). A SIL is normally applied to a subsystem; that is, we can operate with various degrees of SILs within a single system, or within a system of systems. The determination of the SIL for a system under testing is based on a risk analysis.

The standards concerning safety critical systems deal with both development processes and supporting processes, that is, project management, configuration management, and product quality assurance.

Ex.

We take as an example the CEI/IEC 61508 recommends the usage of test case design techniques depending on the SIL of a system. This standard defines four integrity levels: SIL4, SIL3, SIL 2, and SIL1, where SIL4 is the most critical.

For a SIL4-classified system, the standard says that the use of equivalence partitioning is highly recommended as part of the functional testing. Furthermore the use of boundary value analysis is highly recommended, while the use of cause-effect graph and error guessing are only recommended. For white-box testing the level of coverage is highly recommended, though the standard does not say which level of which coverage.

The recommendations are less and less strict as we come down the SILs in the standard.

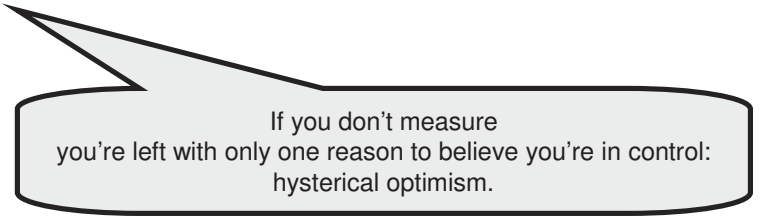
Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 27 2/19/08 8:12:50 PM
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

For highly safety-critical systems the testers may be required to deliver a compliance statement or matrix, explaining how the pertaining regulations have be follow and fulfilled.

1.3 Metrics and Measurement

Tom De Marco, one of the testing gurus, once said:



One of the principles of good planning, both of testing and anything else, is to define specific and measurable goals for the activities. But it is not enough for goals to be measurable; we must also collect facts that can tell us if we have indeed achieved the goals.

1.3.1 Measuring in General

For facts or data collection we operate with the following concepts:

- Metric—A definition of what to measure, including data type, scale, and unit
- Measuring method—The description of how we are going to get the data
- Measurements—The actual values collected for the metrics

An example could be that the metric for the size of a book is "number of pages"; the measuring method is to "look at the last page number"; and the measurement for *Alice in Wonderland,* ISBN 7409746, is "54."

It is a good idea to establish a measurement plan as part of the project plan or master test plan. This should specify the metrics we want to measure and the measuring methods, who is going to measure, and perhaps most importantly: how the measurements will be analyzed and used.

Our measurements are derived from raw data such as time sheets, incident reports, test logs, and work sheets. *Direct measurements* are measurements we get directly from the raw data, for example, by counting the number of log sheets for passed test procedures and counting the number of incident reports. *Indirect measurements* are measurements we can calculate from direct measurements.

Copyright © 2008. Artech House. All rights reserved.

Book-sample.indb 28 2/19/08 8:12:51 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Most direct measurements have no meaning unless they are placed in relation to something. Number of incidents as such—for example, 50—says nothing about the product or the processes. But if we calculate the defects found compared to the estimated amount of defects it gives a much better indication—either of our estimation or of the quality of the product!

It is a common mistake to think that only objective data should be used. Objective data is what you can measure independently of human opinions. But even though subjective data has an element of uncertainty about it, it can be very valuable. Often subjective data is even cheaper to collect than objective data.

A subjective metric could be:

The opinion of the participants in walk-throughs concerning the usefulness of the walk-through activity on a scale from 1 to 5, where 1 is lowest and 5 is highest.

This is easy to collect and handle, and it gives a good indication of the perception of the usefulness of walk-throughs.

The metrics should be specified based on the goals we have set and other questions we would like to get answers to, such as how far we are performing a specific task in relation to the plan and expectations.

1.3.2 Test-Related Metrics

Many, many measurements can be collected during the performance of the test procedures (and any other process for that matter). They can be divided into groups according to the possibilities for control they provide. The groups and a few examples of direct measurements are listed here for inspirational purposes; the lists are by no means exhaustive.

- **Measurements about progress**
  - Of test planning and monitoring:
    - Tasks commenced
    - Task completed
  - Of test development:
    - Number of specified and approved test procedures
    - Relevant coverages achieved in the specification, for example, for code structures, requirements, risks, business processes
    - Other tasks completed
  - Of test execution and reporting:
    - Number of executed test procedures (or initiated test procedures)
    - Number of passed test procedures

Copyright © 2008. Artech House. All rights reserved.

Book.samfer.indb 29 2/19/08 8:12:51 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

- Number of passed confirmation tests
- Number of test procedures run as regression testing
- Other tasks completed
- Of test closure:
  - Tasks completed

For each of these groups we can collect measurements for:

- Time spent on specific tasks both in actual working hours and elapsed time
- Cost both from time spent and from direct cost, such as license fees

- **Measurements about coverage:**
  - Number of coverage elements covered by the executed test procedures code structures covered by the test

- **Measurements about incidents:**
  - Number of reported incidents
  - Number of incidents of different classes, for example, faults, misunderstandings, and enhancement requests
  - Number of defects reported to have been corrected
  - Number of closed incident reports

- **Measurements about confidence:**
  - Subjective statements about confidence from different stakeholders

All these measurements should be taken at various points in time, and the time of the measuring should be noted to enable follow-up on the development of topics over time, for example the development in the number of open incident reports on a weekly basis.

It is equally important to prepare to be able to measure and report status and progress of tasks and other topics in relation to milestones defined in the development model we are following.

To be able to see the development of topics in relation to expectations, corresponding to factual and/or estimated total numbers are also needed. A few examples are:

- Total number of defined test procedures
- Total number of coverage elements
- Total number of failures and defects
- Actual test object attributes, for example, size and complexity
- Planned duration and effort for tasks
- Planned cost of performing tasks

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 30 2/19/08 8:12:52 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

1.3.3 Analysis and Presentation of Measurements

It is never enough to just collect measurements. They must be presented and analyzed to be of real value to us. The analysis and presentation of measurements are discussed in Section 3.4.2.

1.3.4 Planning Measuring

It is important that stakeholders agree to the definition of the metrics and measuring methods, before any measurements are collected. Unpopular or adverse measurements may cause friction, especially if these basic definitions are not clear and approved. You can obtain very weird behaviors by introducing measurements!

There is some advice you should keep in mind when you plan the data you are going to collect. You need to aim for:

- **Agreed metrics**—Definitions (for example, what is a line of code), scale (for example, is 1 highest or lowest), and units (for example, seconds or hours) must be agreed on and understood
- **Needed measures**—What is it you want to know, to monitor, and to control?
- **Repeatable measurements**—Same time of measure and same in strument must give the same measurement
- **Precise measurements**—Valid scale and known source must be used
- **Comparable measurements**—For example, over time or between sources
- **Economical measurements**—Practical to collect and analyse com pared to the value of the analysis results
- **Creating confidentiality**—Never use measurements to punish or award individuals
- **Using already existing measurements**—Maybe the measurements just need to be analyzed in a new way
- **Having a measurement plan**—The plan should outline what, by whom, when, how, why
- **Using the measurements**—Only measure what can be used immediately and give quick and precise feedback

Questions

1. What is the development life cycle and the product life cycle?
2. What are the building blocks in software development models?
3. What are the basic development model types?

Copyright © 2008. Artech House. All rights reserved.

Book: samfer.indb 31 2/19/08 8:12:52 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

4. What is the difference between a waterfall development model and a V-model?
5. What are the advantages of a W-model?
6. What is the main difference between iterative development and incremental development?
7. What are the characteristics of projects following an iterative model?
8. What does RAD stand for?
9. What is the principle in Boehm's spiral model?
10. What are the value principles for agile development?
11. What is the most interesting aspect of XP from a testing point of view?
12. What is the standard V-model that everybody should follow?
13. What is the test object in component testing?
14. What are stubs and drivers used for?
15. When should an individual component test stop?
16. What are the test objects in integration testing?
17. Which integration strategies exist?
18. Which techniques could be used in system testing?
19. How is acceptance testing different from the other test levels?
20. What are the supporting processes discussed in this book?
21. What are the four quality-assurance activities?
22. What is validation?
23. Why is gold-plating dangerous?
24. What is verification?
25. What are the five project management activities?
26. What are the four configuration management activities?
27. What are the purposes of each of them?
28. What can be placed under configuration management from a testing point of view?
29. What is the interface between testing and technical writers?
30. What is a system of systems?
31. What should be considered when testing a system of systems?
32. What are the value categories in safety critical systems?
33. What is a SIL?
34. How can standards guide the testing of a safety-critical system?
35. What is the difference between a direct and an indirect measurement?
36. Why can subjective measurements be useful?
37. What are the groups for which testers can collect information?
38. When should measuring take place?
39. How should measurements be used?
40. What is the most important aspect of measurements apart from using them?

Copyright © 2008. Artech House. All rights reserved.

Book: samlet.indb 32
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:12:52 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

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.

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 33 2/19/08 8:12:53 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

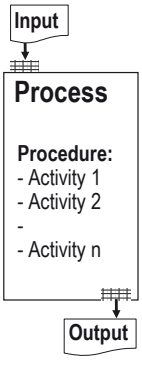
2.1 Processes in General

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.

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 34
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:12:53 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

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.

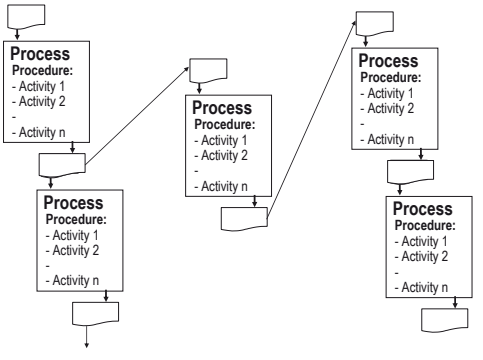
2.1.3 Processes Depend on Each Other

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.

Copyright © 2008. Artech House. All rights reserved.

Book: samfer.indb 35
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:12:54 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

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.

Copyright © 2008. Artech House. All rights reserved.

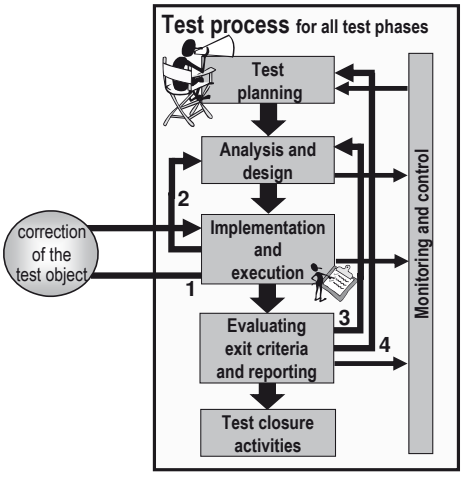
Book1.samlet.indb 36 2/19/08 8:12:54 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

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.

Copyright © 2008. Artech House. All rights reserved.

Book: seiffert.indb 37
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.
2/19/08 8:12:55 PM

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.

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 38 2/19/08 8:12:55 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

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.

Copyright © 2008. Artech House. All rights reserved.

Book-sample.indb 39 2/19/08 8:12:56 PM
Melte, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

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-

Copyright © 2008. Artech House. All rights reserved.

Book-sample.indb 40 2/19/08 8:12:56 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

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.

Copyright © 2008. Artech House. All rights reserved.

Book: samfer.indb 41
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:12:56 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

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

Copyright © 2008. Artech House. All rights reserved.

Book1.sam1et.indb 42 2/19/08 8:12:57 PM
Mette Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

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.

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 43 2/19/08 8:12:57 PM
Mette Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

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)

Copyright © 2008. Artech House. All rights reserved.

Book1.samp.indb 44 2/19/08 8:12:57 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

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.

Copyright © 2008. Artech House. All rights reserved.

Book-sample.indb 45 2/19/08 8:12:58 PM
Meltz, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

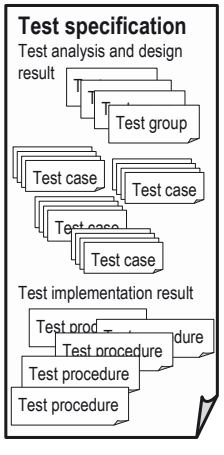
Typical deliveries or work products from a test level are:

- 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

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 46 2/19/08 8:12:58 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

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

2.2.3.5 Scoping the Test Effort

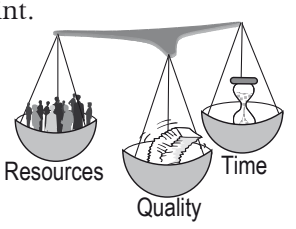
The definition of exhaustive testing is: test case design technique in which the test case suites comprise all combinations of input values and preconditions for component variables. No matter how much we as testers would like to do the ultimate good job, *exhaustive testing is a utopian goal.*

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

Copyright © 2008. Artech House. All rights reserved.

Book: seiffert.indb 47
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:12:59 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

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.

Copyright © 2008. Artech House. All rights reserved.

Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Book-sample.indb 48 2/19/08 8:12:59 PM

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.

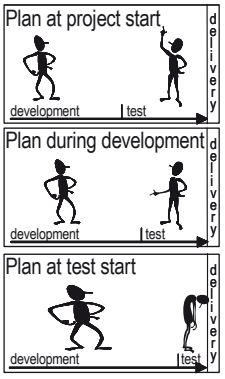
Estimations for all the tasks are input to the scheduling. Once the tasks are estimated they can be fitted into the project time line. Test estimation is discussed in detail in Section 3.3.

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

Copyright © 2008. Artech House. All rights reserved.

Book: samfer.indb 49
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:13:00 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

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

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 50
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:13:01 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

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.

Copyright © 2008. Artech House. All rights reserved.

Book: seamless.indb 51
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:13:01 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

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.

Copyright © 2008. Artech House. All rights reserved.

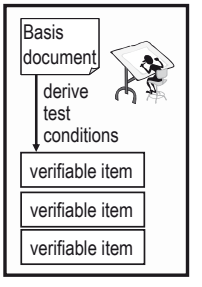
Book: samfet.indb 52
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:13:02 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

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

Copyright © 2008. Artech House. All rights reserved.

Book.sample.indb 53 2/19/08 8:13:02 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

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!

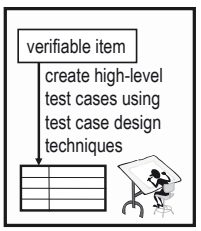
Only if the test conditions are not clearly defined in the basis documentation do we have to document them ourselves. If we do so we must *get the test conditions reviewed* and approved by the stakeholders.

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.

Copyright © 2008. Artech House. All rights reserved.

Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Book: seminet.indb 54 2/19/08 8:13:03 PM

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.

Copyright © 2008. Artech House. All rights reserved.

Book1.sample.indb 55 2/19/08 8:13:03 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

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.

Copyright © 2008. Artech House. All rights reserved.

Book-sample.indb 56 2/19/08 8:13:03 PM
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

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.

Copyright © 2008. Artech House. All rights reserved.

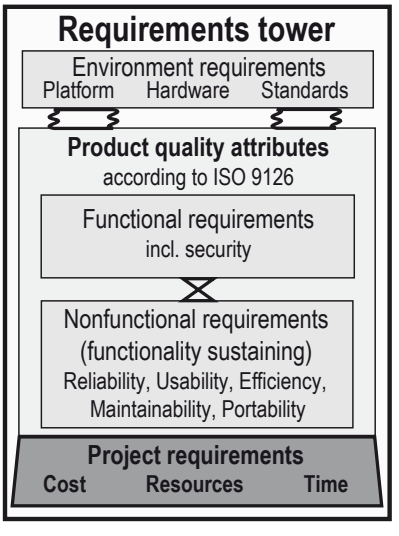
Book: seamet.indb 57
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.
2/19/08 8:13:04 PM

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

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 58 2/19/08 8:13:04 PM
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

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 nonfunctional requirements!

Ex.

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.

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 59
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.
2/19/08 8:13:04 PM

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.

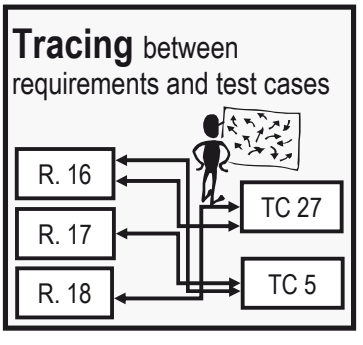
There are two sets of references:

- 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.

Copyright © 2008. Artech House. All rights reserved.

Book.samfm.indb 60 2/19/08 8:13:05 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

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)

Ex.

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.

I

The inputs on which this process is based are:

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

P

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

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 61 2/19/08 8:13:05 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

- 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.

Copyright © 2008. Artech House. All rights reserved.

Book-samler.indb 62 2/19/08 8:13:05 PM
Mette Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

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.

Copyright © 2008. Artech House. All rights reserved.

Book: seiler.indb 63
Mette Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:13:06 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

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

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 64
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:13:06 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

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)

Copyright © 2008. Artech House. All rights reserved.

Book-sample.indb 65 2/19/08 8:13:07 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

- 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.

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 66 2/19/08 8:13:07 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

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

Copyright © 2008. Artech House. All rights reserved.

Book: seminet.indb 67
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:13:07 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

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

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 68 2/19/08 8:13:07 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

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.

Copyright © 2008. Artech House. All rights reserved.

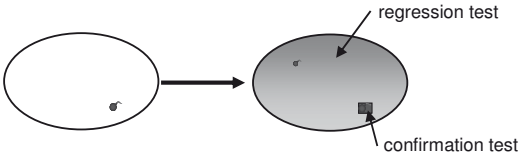
Book: samfer.indb 69 2/19/08 8:13:08 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

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:

Copyright © 2008. Artech House. All rights reserved.

Book-sample.indb 70 2/19/08 8:13:08 PM
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

"… 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.

Ex.

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

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 71 2/19/08 8:13:09 PM
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

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.

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 72
Melle, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:13:09 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

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:

Copyright © 2008. Artech House. All rights reserved.

Book1.sam1e.indb 73 2/19/08 8:13:09 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

- 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.

Copyright © 2008. Artech House. All rights reserved.

Book: seamless.indb 74
Melie, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:13:09 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

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.

Copyright © 2008. Artech House. All rights reserved.

Book-sample.indb 75
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:13:10 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

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?

Copyright © 2008. Artech House. All rights reserved.

Book: samfm.indb 76 2/19/08 8:13:10 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

31. What should be in a test procedure according to IEEE 829?
32. What are the guidelines for the length of a test procedure?
33. Why should test specifications be reviewed?
34. How can the test environment jeopardize the test?
35. What characterizes a valid test environment?
36. Why should test entry criteria be checked?
37. Which supporting process is it especially important to have in place before test execution starts, and why?
38. Why should the test specification be followed during test execution?
39. What must be done when a failure is observed?
40. What information should be recorded for each executed test procedure?
41. What are confirmation testing and regression testing?
42. How much regression testing should be done?
43. When should regression testing be performed?
44. How should test progress and completion reporting be done?
45. Why should we check for completion?
46. What can be done if the completion criteria are not met?
47. What are the activities in the test closure process?
48. Why should testware be kept?
49. What is done in a retrospective meeting?
50. What is the ultimate purpose of the experience report?

Copyright © 2008. Artech House. All rights reserved.

Book1.sample.indb 77 2/19/08 8:13:10 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Copyright © 2008. Artech House. All rights reserved.

Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

CHAPTER

3

Test Management

Contents

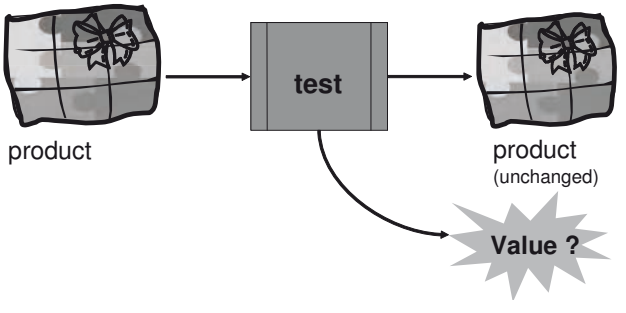
Test management is the art of planning and directing a test assignment to success. It is in many ways like project management, and yet not quite the same.

Test management must be done in close cooperation with project management, sometimes by the same person, sometimes by different people.

The test manager is the link between the test team and the development team and between the test team and higher management. It is therefore essential that the test manager is the ambassador of testing and truly understands how testing contributes to the business goals.

3.1 Business Value of Testing

On the face of it, testing adds no value. The product under testing is—in principle—not changed after the test has been executed.



But we are paid to test, so we must add some value to be in business. And we do!

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 79 2/19/08 8:14:03 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

The business value of testing lies in the savings that the organization can achieve from improvements based on the information the testing provides.

Improvements can be obtained in three places:

- The product under development
- The decisions to be made about the product
- The processes used in both testing and development

It may, however, sometimes be difficult to understand and express what the value is, both to ourselves and to others in the organization.

It is essential that test managers know and understand the value of testing and know how to express it to others to make them understand as well. Test managers must communicate the value to the testers, to other project participants, and to higher management.

Testers are often engrossed in the testing tasks at hand and don't see the big picture they are a part of; higher management is often fairly remote from the project as such and doesn't see the detailed activities.

3.1.1 Purpose of Testing

What testing does and therefore the immediate *purpose of testing is getting information about the product under testing.* We could say (with Paul Gerrard, founder of Aqastra): Testing is the intelligence office of the company.

The places we gather our raw data from are the test logs and the incident reports, if these are used sensibly and updated as the testing and the incident are progressing. From the raw data we can count and calculate a lot of useful quantitative information.

A few examples of such information are:

- Number of passed test cases
- Coverage of the performed test
- Number and types of failures
- Defects corrected over time
- Root causes of the failures

Most of this information is "invisible" or indigestible unless we testers make it available in appropriate formats. There is more about this in Section 3.5. This section also discusses how the information can be used to monitor the progress of the development in general and the testing in particular.

Copyright © 2008. Artech House. All rights reserved.

Book_samler.indb 80 2/19/08 8:14:03 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

3.1.2 The Testing Business Case

It is not straightforward to establish a business case for testing, since we don't know in advance what savings we are going to enable. We don't know how many defects in the product we are going to unveil.

A well-established way to express the value of testing for the product is based on the cost of quality. This can be expressed as value of product improvement:

Value of product improvement =
(cost of failure not found – cost failure found) – cost of detection
To this we can add
Value of decision improvement =
(cost of wrong decision – cost of right decision) – cost of getting decision basis
Value of process improvement =
(cost using old process – cost using better process) – cost of process improvement

These three aspects add up to form the entire business case for testing. The aim is to get as high a value as possible.

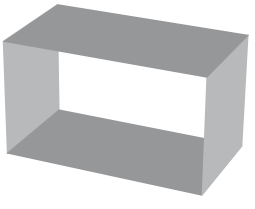
A value may be expressed either quantitatively or qualitatively. Quantitative values can be expressed in actual numbers—euros, pounds, or dollars or numbers of something, for example. Qualitative values cannot be calculated like that, but may be expressed in other terms or "felt."

3.1.2.1 The Value of Product Improvement

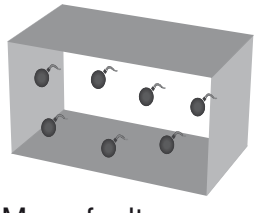
The value of product improvement is the easiest to assess.

One goal of all development is reliability in the products we deliver to the customers. Reliability is the probability that software will not cause the failure of a system for a specified time under specified conditions.

A product's reliability is measured by the probability that faults materialize in the product when it is in use.



No faults
= 100% reliability



Many faults
= *x*% reliability

Copyright © 2008. Artech House. All rights reserved.

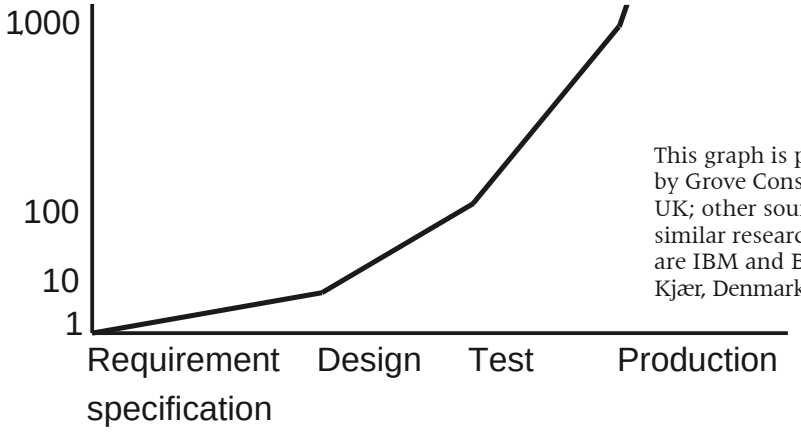
Book: seminet.indb 81 2/19/08 8:14:04 PM
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebook Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

The less failures that remain in the product we release, the higher is the reliability of the product and the lower the risk of the product failing and thereby jeopardizing its environment. Project risks range from ignorable to endangering the lives of people or companies. There is more about risk management in Section 3.6.

The earlier we get a defect removed the cheaper it is. Reviews find defects and dynamic testing finds failures, and this enables the correction of the underlying defects.

The cost of the defect correction depends on when the defect is found. Defects found and corrected early are much cheaper to correct than defects found at a later point in time. Research shows that if we set the cost of correcting a defect found in the requirements specification to 1 unit, then it will cost 10 units to make the necessary correction if the defect is first found in the design.

If the defect remains in the product and is not found until encountered as a failure in dynamic test, it costs 100 units to correct it. The failures found during development and testing are called internal failures, and they are relatively cheap.



This graph is provided by Grove Consultants, UK; other sources for similar research results are IBM and Brüel & Kjær, Denmark.

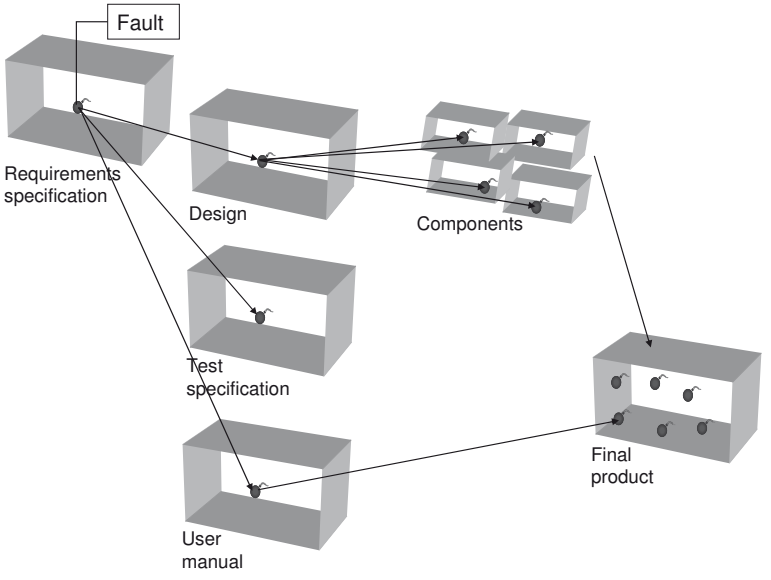
If the customer gets a failure in production—an external failure—it may costs more than 1,000 units to make the necessary corrections, including the cost that the customer may incur. The analysts and programmers who can/must correct the defects may even be moved to new assignments, which are then in turn delayed because of (emergency) changes to the previous product.

The basic reason for this raise in cost is that defects in software do not go away if left unattended; they multiply. There are many steps in software

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 82 2/19/08 8:14:04 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

development from requirements specification to manufacturing and for each step a defect can be transformed into many defects.



There is some element of estimation in preparing the business case for product improvement. Many organizations don't know how many defects to expect, how much it costs to find defects, and how much it costs to fix them, or how much it would have cost to fix them later. The more historical data about the testing and defect correction an organization has, the easier it is to establish a realistic business case.

Ex.

Let's look at a few calculation examples.

If we assume that it costs 4 units to correct a defect in the requirements phase, and 6 units to detect a defect or a failure, we can make calculations like:

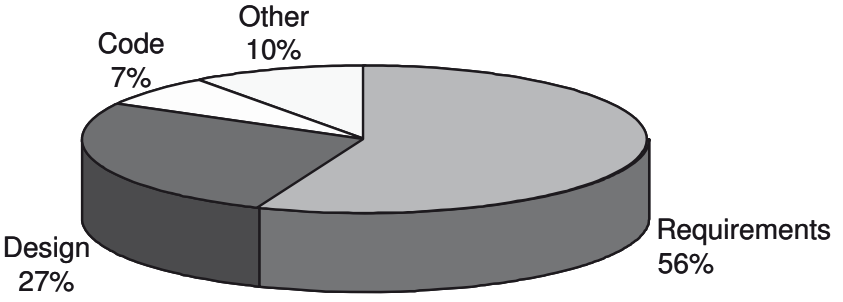
Value of finding a defect in system testing rather than in production at the customer's site = (4,000 – 400) – 6 = 3,594 units

Value of finding a defect in requirements specification rather than in system testing = (400 – 4) – 6 = 390 units

Other research show that about 50% of the defects found in the entire life of a product can be traced back to defects introduced during the requirements specification work. This is illustrated in the following figure where the origins of defects are shown.

Copyright © 2008. Artech House. All rights reserved.

Book: samfer.indb 83 2/19/08 8:14:05 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.



If we combine these two pieces of research results we have a really strong case for testing, and for starting testing early on in the project!

To get the full value of the test, it should start as early as possible in the course of a development project, preferably on day 1!

3.1.2.2 The Value of Decision Improvement

From the point of view of decision making such as decisions concerning release (or not) of a product the confidence in the product and quality of the decisions are proportional to the quality and the amount of the information provided by testing. As testing progresses, more and more information is gathered and this enhances the basis for the decisions.

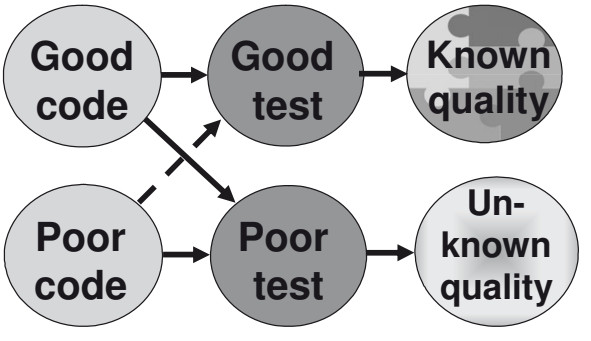
The more knowledge the decision makers have about what parts of the product have been tested to which depth—coverage—and which detected defects have been removed and which are still remaining, the more informed are the decisions made. The value of more informed decisions rather than less informed decisions is qualitative; it is very rarely possible to calculate this quantitatively.

It follows from the concept of test as an information collection activity that it is not possible to test good quality into a product. But the quality of the testing reflects in the quality of the information it provides.

Good testing provides trustworthy information and poor testing leave us in ignorance.

If the starting point is a good product, a good test will provide information to give us confidence that the quality is good.

If the starting point is a poor product, a good test will reveal that the quality is low.



Copyright © 2008. Artech House. All rights reserved.

Book1.sam1et.indb 84 2/19/08 8:14:05 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

But if the testing is poor we will not know if we have a good or a poor product.

The line from "poor code" to "good test" is dashed, because poor coding and good testing is not often seen together. Our goal as professional test practitioners is to reduce the occurrence of poor testing.

More important decisions may also be based on the information from testing. A test report with documentation of the test and the test results can be used to prove that we fulfilled contractual obligations, if needed. It may even in some (one hopes rare) cases provide a judicial shield for the company in that it provides evidence against suits for negligence or the like. This is of qualitative value to the business.

3.1.2.3 The Value of Process Improvement

From *the process improvement* point of view the information gained from testing is invaluable in the analysis of how well processes fit and serve the organization. The results of such analysis can be used to identify the process that could be the subject for process improvement. The process to improve may be both the testing process and other processes.

As time goes by the information can tell us how a process improvement initiative has worked in the organization.

When the testing process improves, the number of failures sent out to the customers falls, and the organization's reputation for delivering quality products will rise (all else being equal). The value of this is qualitative.

There is more about process improvement in Chapter 8.

3.2 Test Management Documentation

3.2.1 Overview

Proper test management requires that information about the decisions that test management makes is available and comprehensive to all stakeholders. These decisions are normally captured in a number of documents.

The test management documentation comprises:

- Test policy
- Test strategy
- Project test plan
- Level test plan

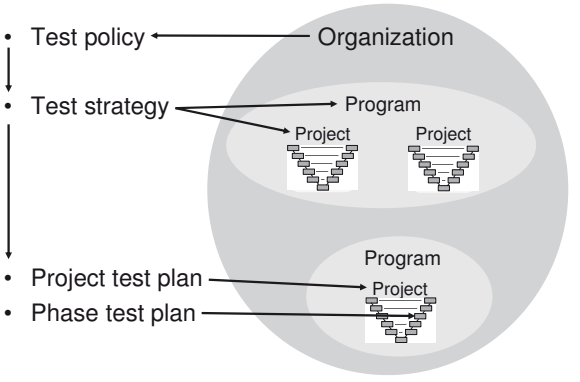
The test management documentation belongs to different organizational levels as shown in the next figure.

Copyright © 2008. Artech House. All rights reserved.

Book: samfer.indb 85
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:06 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

The *test policy* holds the organization's philosophy toward software testing.

The *test strategy* is based on the policy. It can have the scope of an organizational unit or a program (one or more similar projects). It contains the generic requirements for the test for the defined scope.



A *master test plan* is for a particular project. It makes the strategy operational and defines the test levels to be performed and the testing within those levels.

A *level test plan* is for a particular test level in a particular project. It provides the details for performing testing within a level.

The presentation of this documentation depends on the organization's needs, general standards, size, and maturity. The presentation can vary from oral (not recommended!) over loose notes to formal documents based on organizational templates. It can also vary from all the information being presented together in one document, or even as part of a bigger document, to it being split into a number of individual documents.

The more mature an organization is the more the presentation of the test management documentation is tailored to the organization's needs. The way the information is presented is not important; the information is.

3.2.2 Higher Management Documentation

Higher management, that is management above project managers and test managers, is responsible for the two types of test management documentation discussed in this section. The documentation is used by everybody in the organization involved in testing.

3.2.2.1 Test Policy

The test policy defines the organization's philosophy toward software testing. It is the basis for all the test work in the organization. A policy must be behavior-regulating in the good way—*it is like a lighthouse for all the testing activties*. And like every lighthouse has its own signal, every organization must have its own policy, tailored to its specific business needs.

The test policy must be short and to the point.

It is the responsibility of the top management to formulate the policy. It may, however, be difficult for top management if managers are not familiar

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 86
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:07 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

with professional testing, so it is often seen that the IT department (or equivalent) steps in and develops the test policy on behalf of the management.

The test policy must include:

1. Definition of testing
2. The testing process to use
3. Evaluation of testing
4. Quality targets
5. Approach to test process improvement

The test policy applies to all testing. The policy must cover all test targets. This means that there must be a policy for:

- Testing new products
- Change-related testing
- Maintenance testing

Test Policy; Definition of Testing

The definition of testing is a brief statement formulating the overall purpose of the test in the organization.

"Checking that the software solves a business problem"
"Activity to provide information about the quality of the products"
"A tool box for minimization of the product risks"

Test Policy; The Testing Process

The testing process is an overview of the activities to be performed or a reference to a full description of the testing process.

"Development and execution of a test plan in accordance with departmental procedures and user requirements found on the intranet"

Another possibility is a reference to the test process defined in standards or other literature, for example the ISTQB syllabus, the test process on which this book based. This test process was described in detail in Chapter 2.

Test Policy; Evaluation of Testing

The evaluation of testing is the measurement to be made in order for the quality of the testing to be determined.

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 87
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:07 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

"The number of failures reported by the field is measured every three months."
"The cost of the fault correction done after release is measured."
"The customer satisfaction is measured once a year by means of a questionnaire sent out to 200 selected customers."

Test Policy; Quality Targets

The quality targets to be achieved should be expressed so that the measurements can be used to see if we reach the goals.

Examples are:

"No more than one high severity fault per 1,000 lines of delivered code to be found in the first six months of operation."

"The overall effectiveness of the test must be over 98% after the first three months in production."

"The customers must not be reporting more than three severity 1 failures during the first year of use."

"The system must not have a breakdown lasting longer than 15 minutes during the first six months in production."

Test Policy; Approach to Test Process Improvement

The organizational approach to test process improvement is the process to be used for learning from experiences. This would often be the same as the organization's general approach to software process improvement, but there might be a specific policy for the testing process improvement.

"A postproject workshop where all the observations during the test process are collected shall be held within the first month after turnover to production."

"Failure reports shall be analyzed to determine any trends in the faults found in system test."

"The root cause shall be found for every severity 1 and 2 fault found during testing, and improvement actions shall be determined."

3.2.2.2 Test Strategy

The Latin word "stratagem" means a plan for deceiving an enemy in war. The enemy here is not the developers, but rather the defects! The strategy is based on the test policy and should of course be compliant with it. The strategy outlines how the risks of defects in the product will be fought. It could be said to express the generic requirements for the test. The strategy comprises the navigation rules for the testing.

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 88
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:07 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

The test strategy is high-level, and it should be short. It should also be readily available to all with a stake in the testing within the scope of the strategy. The strategy could be issued in a document, but it would be a good idea to present it in table form on a poster or on the intranet in the organization.

A test strategy must be for a specified scope. The scope may be all the projects in an entire organization, a specific site or department, or a program (a number of similar projects).

The overall test strategy may be chosen among the following possible approaches to the testing:

- Analytical—Using for example a risk analysis as the basis
- Model-based—Using for example statistical models for usage
- Consultative—Using technology guidance or domain experts
- Methodical—Using for example checklists or experience
- Heuristic—Using exploratory techniques
- Standard-compliant—Using given standards or processes
- Regression-averse—Using automation and reuse of scripts

There is nothing wrong with mixing the approaches. They address different aspects of testing and more approaches can support each other.

We could, for example, decide:

The component test shall be structured in compliance with the tool used for component testing.

The component integration testing shall be bottom-up integration based on a design model and in compliance with standard xxx.

The system test shall be risk-based and structured and the initial risk analysis shall be supplemented with exploratory testing.

The decisions about approaches or overall strategies have a great influence on some of the decisions that have to be made for specific topics in the strategy.

Remember that the strategy must be short—the test approach is to be refined and detailed in the test plans.

The test strategy should not be "once-written-never-changed." As the experiences gained from finished testing activities are collected and analyzed, the results in terms of test process improvement initiatives must constantly be considered when the test strategy is formulated.

Copyright © 2008. Artech House. All rights reserved.

Book-sample.indb 89 2/19/08 8:14:07 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

A test strategy for a defined scope could contain the following information:

Test strategy identifier

1. Introduction
2. Standards to use
3. Risks to be addressed
4. Levels of testing and their relationships
   For each level, as appropriate
   - 4.1 Entry criteria
   - 4.2 Exit criteria
   - 4.3 Degree of independence
   - 4.4 Techniques to use
   - 4.5 Extent of reuse
   - 4.6 Environments
   - 4.7 Automation
   - 4.8 Measurements
   - 4.9 Confirmation and regression testing
5. Incident management
6. Configuration management of testware
7. Test process improvement activities

Approvals

The numbered topics are indented as sections in the strategy. The identifier and the approvals are information about the strategy usually found on the front page.

The strategy identifier is the configuration management identification information for the strategy itself. It could be formed by the:

- Name of the strategy
- Organizational affiliation
- Version
- Status

This should adhere to the organization´s standards for configuration management, if there is one.

Strategy; Introduction

The introduction sets the scene for the strategy. It contains general information of use to the reader.

The introduction is usually the most organization specific chapter of the plan. It should be based on the organization's own standard. It should in any case cover:

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 90
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:08 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

- Purpose of the document
- Scope of the strategy
- References to other plans, standards, contracts, and so forth
- Readers' guide

Strategy; Standards to Be Complied With

In this section references to the standard(s) that the test must adhere to are provided.

Standards may be both external to the organization and proprietary standards.

Standards are very useful. Many people with a lot of experience have contributed to standards. Even though no standard is perfect and no standard fits any organization completely standards can facilitate the work by providing ideas and guidelines.

The more standards it is possible to reference the easier the work in the strategy, the planning, and the specification. Information given in standards must not be repeated in specific documents, just referenced.

IEEE 829, Test Documentation.
"Test-Nice"—the company standard for test specifications.

Some appropriate standards are discussed in Chapter 8.

Strategy; Risks

The basis for the strategy can be the product risks to mitigate by the testing. Appropriate project risks may also be taken into consideration.

The strategy must include a list of the relevant risks or a reference to such a list.

Risks in relation to testing are discussed in Section 3.8.

Strategy; Test Levels and Their Relationships

The typical test strategy will include a list and description of the test levels into which the test assignments within the scope should be broken.

The levels can for example be:

- Component testing
- Component integration testing
- System testing
- System integration testing
- Acceptance testing

The levels are described in detail in Chapter 1.

Copyright © 2008. Artech House. All rights reserved.

Book: samfer.indb 91
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:08 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

The following strategy topics must be addressed for each of the levels that the strategy includes. It is a good idea to give a rationale for the decisions made for each topic, if it is not obvious to everybody.

Strategy; Level Entry Criteria

This is a description of what needs to be in place before the work in the test level can start.

The strictness of the entry criteria depends on the risk: The higher the risk the stricter the criteria.

An entry criterion for the system test could be that the system requirements specification has passed the first review.

Strategy; Level Exit Criteria

The testing exit or completion criteria are a specification of what needs to be achieved by the test. It is a guideline for when to stop the testing—for when it is "good enough." It is a description of what needs to be in place before the work in the test level can be said to be finished.

Testing completion criteria represent one of the most important items in a comprehensive test strategy, since they have a great influence on the subsequent testing and the quality of a whole system.

Some completion criteria are closely linked to the chosen test case design techniques; some are linked to the progress of the test.

The strategy does however not need to be very specific. The completion criteria will be detailed and made explicit in the test plans. A detailed discussion of completion criteria is found in Section 3.2.3.3.

The strictness of the completion criteria depends on the risk as described above.

Descriptions of the strategy for completion criteria could for example be:

Component test: Decisions coverage must be between 85% and 100% dependent on the criticality of the component. No known faults may be outstanding.

System test: At least 95% functional requirements coverage for priority 1 requirements must be achieved. No known priority 1 failures may be outstanding.

The test report has been approved by the project manager.

Strategy; Degree of Independence

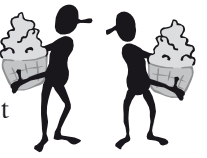
Testing should be as objective as possible. The closer the tester is to the producer of the test object, the more difficult it is to be objective.

The concept of independence in testing has therefore been introduced.

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 92
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:08 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

The degree of independence increases with the "distance" between the producer and the tester. These degrees of independence in testing have been defined:



1. The producer tests his or her own product
2. Tests are designed by another nontester team member
3. Tests are designed by a tester who is a member of the development team
4. Tests are designed by independent testers in the same organization
5. Tests are designed by organizationally independent testers (consultants)
6. Tests are design by external testers (third-party testing)

As it can be seen in the list the point is who designs the test cases. In structured testing the execution must follow the specification strictly, so the degree of independence is not affected by who is executing the test. In testing with little or no scripting, like exploratory testing, the independence must be between producer and test executor.

The strategy must determine the necessary degree of independence for the test at hand. The higher the risk the higher the degree of independence.

There is more about independence in testing in Section 10.4.

Strategy; Test Case Design Techniques to Be Used

A list of the test case design techniques to be used for the test level should be provided here. The choice of test case design techniques is very much dependent on the risk—high risk: few, comprehensive techniques to choose from; low risk: looser selection criteria.

Test case design techniques could be equivalence partitioning, boundary value analysis, and branch testing for component testing.

Test case design techniques are described in great detail in Chapter 4.

Strategy; Extent of Reuse

Reuse can be a big money and time saver in an organization.

Effective reuse requires a certain degree of maturity in the organization. Configuration management needs to be working well in order to keep track of items that can be reused.

This section must provide a description of what to reuse under which circumstances.

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 93
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:09 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

Work product for reuse could, for example, be:

- Generic test specifications
- Specific test environment(s)
- Test data

Strategy; Environment in Which the Test Will Be Executed

Generic requirements for the test environment must be given here. The specific environment must be described in the test plan, based on what the strategy states.

The requirements for the test environment depend very much on the degree of independence and on the test level at which we are working.

We could for example find:

Component testing: The developer's own PC, but in a specific test area.

System test: A specific test environment established on the test company's own machine and reflecting the production environment as closely as possible.

Strategy; Approach to Test Automation

This is an area where the strategy needs to be rather precise in order for tool investments not to get out of hand.

Technical people—including testers—love tools. Tools are very useful and can ease a lot of tedious work.

Tools also cost a lot both in terms of money over the counter and in terms of time to implement, learn, use, and maintain. Furthermore, no single tool covers all the requirements for tool support in a test organization, and only a few tools are on speaking terms. It can be costly and risky, or indeed impossible to get information across from one tool to another.

It is important that the strategy includes a list of already existing testing tools to be used, and/or guidelines for considerations of implementation of new tools.

Test tools are described in Chapter 9.

Strategy; Measures to Be Captured

In the test policy it has been defined how the test shall be evaluated. It has also been defined what the approach to process improvement is. This governs the measures we have to collect.

Measures are also necessary to be able to monitor and control the progress of the testing. We need to know how the correspondence is between the reality and the plan. We also need to know if and when our completion criteria have been met.

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 94 2/19/08 8:14:09 PM
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Based on this, this section must contain a definition of all the metrics for testing activities. Descriptions of metrics include scales, ways of capturing the measurements, and the usage of the collected information.

Metrics and measurements are discussed in general in Section 1.3.

Strategy; Approach to Confirmation Testing and Regression Testing

Confirmation testing is done after fault correction to confirm that the fault has indeed been removed.

Fault correction is NOT part of the test process.

Regression testing should be done whenever something has changed in the product. It is done to ensure that the change has had no adverse effect on something that was previously working OK. Regression testing should follow any confirmation test; it should also for example follow an upgrade of the operation system underlying the product. The amount of regression testing to perform after a change is dependent on the risk associated with the change.

This section must outline when and how to perform confirmation testing and regression testing in the test level it is covering.

System testing: Re-execute the test case(s) that identified the fault and rerun at least 1/3 of the rest of the already executed test cases. The choice of test cases to rerun must be explained.

Strategy; Approach to Incident Management

It is hoped that a reference to the configurations management system is sufficient here.

If this is not the case it must be described how incidents are to be reported and who the incident reports should be sent to for further handling.

Close cooperation with the general configuration management function in the organization is strongly recommended on this. There is no need to reinvent procedures that others have already invented.

Strategy; Approach to Configuration Management of Testware

Configuration management of testware is important for the reliability of the test results. The test specification and the test environment including the data must be of the right versions corresponding to the version of product under testing. A good configuration management system will also help prevent extra work in finding or possibly remaking testware that has gone missing—something that happens all too often in testing.

Configuration management is a general support process, and if a configuration management system is in place this is of course the one the testers should use as well, and the one to which the strategy should refer.

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 95
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:09 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

If such a system is not in place the approach to local testing configuration management must be described. Configuration management is discussed in Section 1.1.3.

Strategy; Approach to Test Process Improvement

This could be a refinement of the approach described in the policy; see Section 3.1.1.5.

3.2.3 Project Level Test Management Documentation

The two types of test management documentation discussed in this section belong to a particular project. The master test plan should be produced by the person responsible for testing on the project, ideally a test manager. The level test plans should be produced by the stakeholder(s) carrying the appropriate responsibility. This could be anybody from a developer planning a component test to the test manager planning the system or acceptance test.

3.2.3.1 Master Test Plan

The plan outlines the journey.

The master test plan documents the implementation of the overall test strategy for a particular project. This is where the strategy hits reality for the first time. The master test plan must comply with the strategy; any noncompliance must be explained.

The master test plan must be closely connected to the overall project plan, especially concerning the schedule and the budget! The master test plan should be referenced from the project plan or it could be an integrated part of it.

The master test plan has many stakeholders and missions, and it must at least provide the information indicated in the following list to the main stakeholders.

Copyright © 2008. Artech House. All rights reserved.

Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Book_samlet.indb 96 2/19/08 8:14:10 PM

Stakeholder	Information
All	Test object = scope of the test for each level
	Involvement in the testing activities
	Contribution to the testing activities
	Relevant testing deliverables (get/produce)
Management	Business justification and value of testing
	Budget and schedule
	Test quantity and quality
Development	Expectation concerning delivery times
	Entry criteria for deliverables
Test team	Test levels
	Schedule
	Test execution cycles
	Suspension criteria and exit criteria
Customer	Test quantity and quality

All stakeholders in the master test plan must agree to the contents according to their interest and involvement—otherwise the plan is not valid!

As mentioned previously, the way the information in the master test plan is presented is not important; the information is.

The detailed structure and contents of a master test plan are discussed in Section 3.2.3.3.

3.2.3.2 Level Test Plan

A level test plan documents a detailed approach to a specific test level, for example a component test or acceptance test. The level test plan describes the implementation of the master test plan for the specific level in even more precise detail. For instance, it would normally include a sequence of test activities, day-to-day plan of activities, and associated milestones.

The size of a level test plan depends on the level it covers—a component test plan for single components may be just 5–10 lines; system test plans may be several pages.

As for the master test plan it is vital to include all relevant stakeholders in the planning process and to get their sign-off on the plan.

3.2.3.3 Test Plan Template

The structure of the test plans, both the master test plan and any level test plans, should be tailored to the organization's needs.

In order not to start from scratch each 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:

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 97
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:10 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

Test plan identifier

1. Introduction (scope, risks, and objectives)
2. Test item(s) or test object(s)
3. Features (quality attributes) to be tested
4. Features (quality attributes) 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
10. Environmental needs
11. Responsibilities
12. Staffing and training needs
13. Schedule
14. Risks and contingencies

Test plan approvals

The numbered topics are intended as sections in the plan; the identifier and the approvals for the plan are usually found on the front page.

The test plan identifier is the configuration management identification information for the test plan itself. It could be formed by the

- Name of the plan
- Organizational affiliation
- Version
- Status

This should adhere to the organization's standards for configuration management, if there is one.

Test Plan; Introduction

The introduction sets the scene for the test plan as a whole. It contains general information of use to the reader.

The introduction is usually the most organization-specific chapter of the plan. It should be based on the organization's own standard. It should in any case cover:

- Purpose of the document
- Scope of the plan, possibly including intended readership
- References to other plans, standards, contracts, and so forth
- Definitions
- Abbreviations

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 98 2/19/08 8:14:10 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

- Typographical conventions used
- Readers' guide

It is important to get the references precise and correct. Test planning is influenced by many aspects, including the organization's test policy, the test strategy, the development or maintenance plan, risks, constraints (time, money, resources), and the test basis and its availability and testability. References must be made to all this information—and it must be respected.

If this chapter gets too voluminous you can place some of the information in appendices.

Test Plan; Test Item(s)

Here the test object(s) or item(s) and additional information are identified as precisely and explicitly as possible. The additional information may be the appropriate source specification, for example detailed design or requirements specification, and helpful information such as the design guide, coding rules, checklists, user manual, and relevant test reports.

The test object depends on whether the plan is the master test plan or a level test plan, and in the latter case of the specific test level the plan is for.

Product XZX V2.3, based on XZX System Requirements Specification V.4.2.
The individual component: pre_tbuly V2.3.

Test Plan; Features to Be Tested

Within the scope of the test item(s), an overview of the features to be tested is provided along with references to where the test is specified, or will be specified as the case may be. Features include both functional and nonfunctional quality attributes.

The decision about which features are to be tested and which are not is based on the applicable test strategy, the identified risks, and the mitigation activities for them. The identification of the features to be tested is also closely linked to the specified coverage items.

All functional requirements, specified in System test specification STS-XX.doc must be covered in this test.

All methods in the classes are to be tested in the component testing.

Test Plan; Features Not to Be Tested

To set the expectations of the stakeholders correctly, it is just as important to state what features are not tested as it is to state which are.

With regards to what might be expected to be tested in relation to the test item(s), we must provide a list of the features not to be tested. A reason must be given for each of the features omitted.

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 99
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:10 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

Performance testing is not part of this test because it will be carried out by third-party company PTESTIT. They are experts in performance testing.

In this component the function M-bladoo is not tested. It is too costly to simulate the error situation that it handles. A formal inspection has been performed on the code.

Test Plan; 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.

Test Plan; Item Pass/Fail Criteria

The item pass/fail criteria are the American counterpart to what we Europeans call 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.

Examples of appropriate completion criteria for some test levels are:

- Component testing
  - 100% statement coverage
  - 95% decision coverage
  - No known faults
- Acceptance testing
  - 100% business procedure coverage
  - No known failures of criticality 1

Test Plan; Suspension Criteria and Resumption Requirements

Sometimes it does not make sense to persevere with the test execution. It can be a very good idea to try to identify such situations beforehand. In this section in the plan, the circumstances that may lead to a suspension of the test for a shorter or longer period are described.

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 100
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:11 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

More than 20% of the time is spent on reporting banal failures, caused by faults that should have been found in an earlier test phase.

It must also be decided and documented what must be fulfilled for the test to be resumed.

Evidence of required coverage of component testing must be provided.

Finally it should be stated which test activities must be repeated at resumption. Maybe every test case must be re-executed; maybe it is OK to proceed from where we stopped.

Test Plan; Test Deliverables

The deliverables are a listing and a brief description of all the documentation, logs, and reports that are going to be produced in the test process at hand. Everything must be included for the purpose of estimation and the setting of expectations.

Example of test deliverables are:

- Test plans
- Test specifications
- Test environment
- Logs, journals, and test reports
- Release documentation for the test object

Test Plan; Testing Tasks

This section in the plan is the work breakdown structure of the test process at hand. If we use the test process used here, it is analysis, design, implementation, execution, evaluation, reporting, and closure, all broken down into more detailed activities in an appropriate work breakdown structure. When defining the test tasks in detail it is important to remember and mention everything. Even the smallest task, which may seem insignificant, may have a significant influence on the schedule.

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

Test Plan; Environmental Needs

The test environment is a description of the environment in which the test is to be executed. It is important to be as specific as possible in order to get the right test environment established at the right time (and at the right cost).

Copyright © 2008. Artech House. All rights reserved.

Book: samfer.indb 101 2/19/08 8:14:11 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Test Plan; Responsibilities

In this section we must describe who is responsible for what. The distribution of testing roles or tasks on organizational units or named people can be shown in a responsibility distribution matrix (RDM). This is a simple two-dimensional matrix or table. On one axis we have organizational units or people, on the other axis we have roles or tasks. In the cross-field we can indicate the type of involvement an organizational unit has for the role.

A completed responsibility distribution matrix might look like this.

	1	2	3	4	5	6
Test leader	R	C	I	I	I	I
Test department	C	R	R	P	P	R
Quality assurance	C	C	R	-	-	I
Sales/marketing	C	C	C	-	-	P
The customer	C	C	C	-	R	P
Method department	I	I	P	R	-	-
Responsible **P**erforming **C**onsulted **I**nformed						

Where:
1. Test management
2. Test analysis and design
3. Test environment
4. Test tools
5. Test data
6. Test execution

Test Plan; Staffing and Training Needs

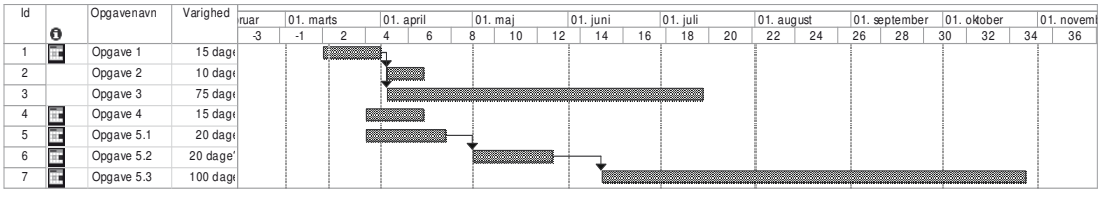
The necessary staff to fulfill the roles and take on the responsibilities must be determined and described here.

Each of the roles requires 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.

Test Plan; Schedule

In the 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.

Scheduling testing is just like any other project scheduling. The result may be presented graphically, typically as Gantt diagrams.



Copyright © 2008. Artech House. All rights reserved.

Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Book1.samlet.indb 102 2/19/08 8:14:12 PM

Test Plan; Risks and Contingencies

This is the management of the risks specifically connected to the task of testing itself, not to the object under test.

The risks to consider here are hence:

- Project risks—What can jeopardize the plan
- Process risks—What can jeopardize the best possible performance of the tasks

The risks must be identified, analyzed, mitigated as appropriate, and followed up like any other risk management task.

Risk management is discussed in Section 3.5.

The *approvals* are the sign-off on the plan by the relevant stakeholders.

3.2.3.4 Scheduling Test Planning

Planning is important and planning takes time.

If you fail to plan—you plan to fail!

It is important to plan activities rather than just jump headfirst into action. The work on the planning provides a deeper understanding of the task at hand, and it is much easier to change something you have written down or sketched out on a piece of paper, than something that has already taken place in the real world.

Because planning takes time and because it is important, it should be planned so that it can start as early as possible. Take your planning seriously, so that you don't end up like this poster painter once did:

The benefits of starting test planning early are many:

- There is time to do a proper job of planning.
- There is more time to talk and/or negotiate with stakeholders.
- Potential problems might be spotted in time to warn all the relevant stakeholders.
- It is possible to influence the overall project plan.

Copyright © 2008. Artech House. All rights reserved.

Book: samfer.indb 103
Mette, Jonassen Haug, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:13 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

When you plan you have to keep in mind that a plan needs to be SMART:

Specific—Make it clear what the scope is
Measurable—Make it possible to determine if the plan still holds at any time
Accepted—Make every stakeholder agree to his or her involvement
Relevant—Make references to additional information; don't copy it
Time-specific—Provide dates

The test plan should be reviewed and approved by all stakeholders to ensure their commitment. A plan is invalid without commitment from the contributors.

Remember that *a plan is just a plan*; it is not unchangeable once written. A plan must be a living document that should constantly be updated to reflect the changes in the real world. Contrary to what many people think it is not a virtue to keep to a plan at any cost—the virtue lies in getting the plan to align with the real world. No matter how hard you try, you are not able to see what is going to happen in the future.

You should always plan *The New Yorker* way: Adjust the detailing of the planning with the visibility at any given moment. When close to an activity provide many details; for activities further away provide fewer details.

As the time for the execution of activities approaches, more details can be provided, and the necessary adjustments done.

All this takes time and it should not be "invisible" work (i.e., work that is not scheduled reported anywhere). The same in fact holds true for the monitoring activities and for the test reporting.

3.2.3.5 Test Report

The purpose of test reporting is to summarize the results and provide evaluations based on these results.

A test report should be issued at the completion of each test level and the end of the entire testing assignment task. The test reports should include analysis of result information to allow management decisions, based on risk, on whether to proceed to the next level of test or to project implementation, or whether more testing is required. Top management may also need information for regularly scheduled project status meetings and at the end of the project in order to adjust policy and strategy.

Test policy
- Definition of testing
- Test process to use
- Test evaluation
- Quality targets
- Test process improvement activities

According to IEEE 829 the test report should contain:

Test report identifier

1. Summary
2. Variances
3. Comprehensiveness assessment
4. Summary of results

Copyright © 2008. Artech House. All rights reserved.

Book: seiffert.indb 104 2/19/08 8:14:14 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

5. Evaluation
6. Summary of activities

Approvals

The *test report identifier* is the identification information for the report itself. As for all the other documentation, it could be formed by the name of the report, the organizational affiliation, the version, and the status.

If the report is to be placed under configuration management, the identification should adhere to the organization's standards for configuration management, if there is one.

Test Report; Summary

The summary provides an overview of test activities. This section could refer to the test plan. The summary should also include any conclusion. It should be possible to read the summary in isolation and get the main information about the test.

Test Report; Variances

The variances to be reported here are all incidents that have happened for any of the items used as a basis for the test. It must also include a summary of what was done and not done with regard to the original plan.

A variance could for example be the issue of a new version of the requirements specification after the approval of the test specification.

Test Report; Comprehensiveness Assessment

In this section we report whether we made it or not according to the original (or modified) plan. It should describe which of the planned tests were not performed, if any, and why not.

This is where we must describe how we met the original completion criteria. If they where modified, this is where we explain why.

Any statistically valid conclusions that can be drawn from these analyses could be used to predict the quality level achieved by the tested product. They can also be used to compare with the target level established in the test plans.

Test Report; Summary of Results

We must provide an overview of incidents found and incidents solved during the testing.

We can also list findings about which functions are working and which functions are not; or about which risks have been eliminated and which are still outstanding.

Copyright © 2008. Artech House. All rights reserved.

Book: samfer.indb 105
Mette, Jonassen Haug, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:14 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

The evaluation sums up the expectations versus the actual findings. Any out-of-scope situations should also be documented as should outstanding issues.

We can draw conclusions regarding the quality of software by comparing the planned quality levels with the actual. We can also give recommendations, but it is not the responsibility of the testers to decide whether the test object should be released or not. That is a management decision—project management, product management, or even higher up.

Test Report; Evaluation

In this section we should give an overall evaluation of the test item, preferably based on a risk analysis of possible outstanding risks related to the release of the item.

The evaluation must be based on the result of the test compared to the completion criteria.

Test Report; Summary of Activities

Here we must provide an overview of the resource usage for the testing. This could be in terms of time used and other costs such as investments in tools.

The *approvals* here are the approvals of the *test report,* not of the test object.

3.3 Test Estimation

3.3.1 General Estimation Principles

Estimation is a prediction of how much time it takes to perform an activity. It is an approximate calculation or judgment, not something carved in stone. An estimate is typically based on the professional understanding of experienced practitioners.

There are many ways in which to express estimations, but the best way is in hours. In that case we don't get problems with holidays, effective working hours, and so forth. You must never express estimates using dates; dates and estimates are incompatible.

Estimation is input to the scheduling. Only in that activity will we transform the estimated hours into dates.

We can also estimate other elements than just time, for example, number of test cases, number of faults to be found, and number of iterations in the test process needed to fulfill the completion criteria. We may also estimate any other costs, such as hardware and tools.

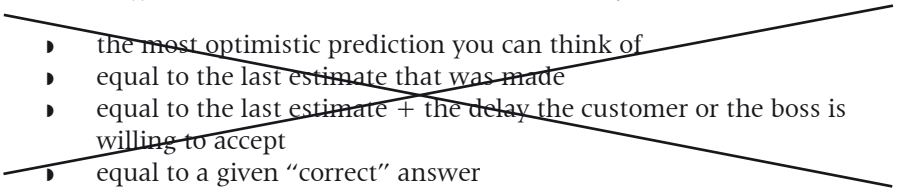
We should always take our estimations seriously. Be honest when you estimate, even though it is often easier to get forgiveness than permission. Keep your original estimates for future reference.

Copyright © 2008. Artech House. All rights reserved.

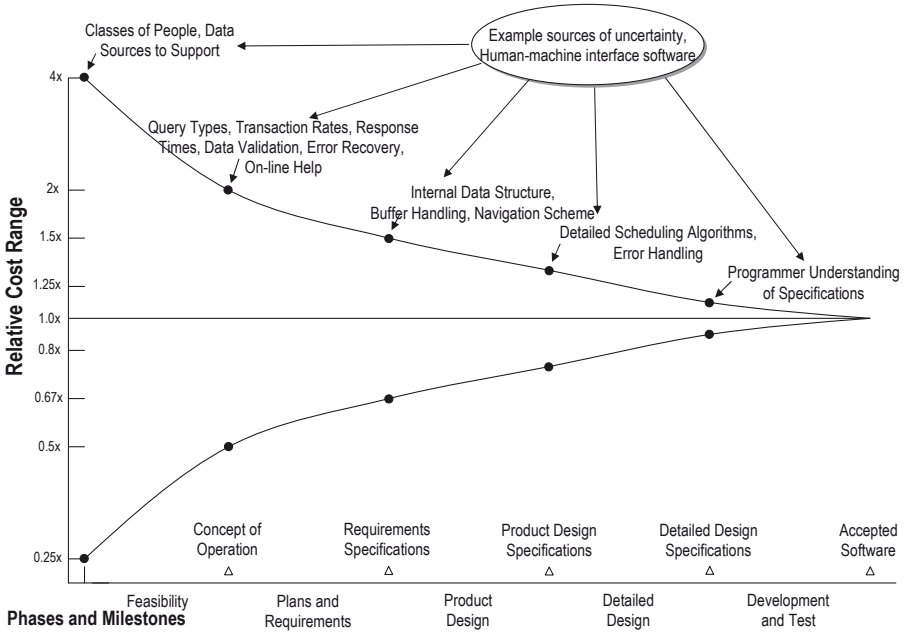
Book: samfet.indb 106 2/19/08 8:14:14 PM
Mette Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

In line with this remember that estimation is not:

- the most optimistic prediction you can think of
- equal to the last estimate that was made
- equal to the last estimate + the delay the customer or the boss is willing to accept
- equal to a given "correct" answer



Estimates are predictions about the future and predictions are by definition uncertain. The closer we come to the actual result, the less is the uncertainty as illustrated here.



You should always calculate with an uncertainty in every estimate and document this uncertainty with the estimate. Furthermore, estimates should always be accompanied by the rationale or justification for the estimation values along with any assumptions and prerequisites.

3.3.2 Test Estimation Principles

Estimating test activities is in many ways like all other estimation in a project. We need to take all tasks, even the smallest and seemingly insignificant, into account.

The time to complete must be estimated for each task defined in the task section, including all the test process activities from test planning to checking for completion.

Copyright © 2008. Artech House. All rights reserved.

Mette, Jonassen Hass. Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Book: samtest.indb 107 2/19/08 8:14:15 PM

Even though estimation of testing tasks is in many ways identical to the estimation for any other process, there are also important differences. A test is a success if it detects faults—this is the paradox with which we have to deal.

The test estimation is different from other project estimations, because the number of failures is not known in advance—though it can be estimated as well. The number of necessary iterations before the completion criteria are met is usually not known either. As a rule of thumb, at least three iterations must be reckoned with—one is definitely not enough, unless the completion criterion is a simple execution of all test cases, and independent of the number of outstanding faults and coverage.

Nevertheless, we have to do our best. The estimation must include:

- Time to produce incident registrations
- Possible time to wait for fault analysis
- Possible time to wait for fault correction
- Time for retest and regression test (minimum three iterations!)

The reason why we have to cater for several iterations is that, well: "*Errare humanum est*!"

When we report incidents and the underlying faults are corrected by development or support staff, not all reported faults are actually corrected. Furthermore, fault correction introduces new faults, and fault correction unveils existing faults that we could not see before.

Experience in the testing business shows that *50% of the original number of faults remains after correction.* These are distributed like this:

Remaining faults after correction	20%
Unveiled faults after correction	10%
New faults after correction	20%

So if we report 100 faults, we have 20 + 20 + 10 = 50 faults to report in the next iteration, 10 + 10 + 5 = 25 faults in the third, and 5 + 5 + 2 = 12 in the forth.

These are general experience numbers. It is important that you collect your own metrics!

3.3.3 The Estimation Process

Estimation is a process like anything else we do. You should of course use your organization's standard process for estimation, if there is one. Otherwise, you can adapt an estimation procedure like the generic one described here.

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 108 2/19/08 8:14:15 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

1. Define the purpose of the estimation—Is this estimation the first approach, for a proposal, or for detailed planning?
2. Plan the estimating task—Estimation is not a two-minute task; set sufficient time aside for it.
3. Write down the basis for the estimation—Here the scope and the size of the work are determined, and all factors that may influence the estimates are registered. This includes factors related to the nature of the processes we are working by, the nature of the project we are working in, the people we are working with, and any risks we are facing.
4. Break down the work—This is the work breakdown (i.e., the listing of all the tasks to estimate). Do this as well as possible in relation to the purpose.
5. Estimate—Use more than one technique as appropriate.
6. Compare with reality and reestimate—This is the ongoing monitoring and control of how the work that we have estimated is actually going.

3.3.4 Estimation Techniques

The following estimation techniques are the most used and an expression of the best practice within estimation.

- FIA (finger in the air) or best guess
- Experience-based estimation
- Analogies and experts
- Delphi technique
- Three-point estimation (successive calculation)
- Model-based estimation
- Function points
- Test points
- Percentage distribution

3.3.4.1 Estimation; Best Guess (FIA)

This technique is more or less pure guesswork, but it will always be based on some sort of experience and a number of (unconscious) assumptions. The technique is very widespread, but since it is based on your gut feeling it is bound to be inaccurate. It is often not repeatable, and it is not always trusted.

The uncertainty contingency is probably around 200%–400% for estimates based on best guess. We can do better than that.

Copyright © 2008. Artech House. All rights reserved.

Book: samfer.indb 109
Mette, Jonassen Haug, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:16 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

3.3.4.2 Estimation; Analogies and Experts

In the analogy techniques you base your estimate on something you have experienced before.

For example: "This looks very much like the system I tested in my previous job. That took us three months, and we were four people. This is slightly smaller and we are five people—so I guess this will take two months to complete."

If you have participated in a testing project that is comparable to the one you are estimating, you might use that as a baseline to do your estimation.

Analogies may also be based on metrics collected from previous tests. We may estimate the number of iterations of the test based on recent records of comparable test efforts. We can calculate the average effort required per test on a previous test effort and multiply by the number of tests estimated for this test effort.

Experts, in the estimation context, know what they are talking about and have relevant knowledge. It is almost always possible to find experts somewhere in the organization.

If experts on this kind of testing are available, then by all means make use of them. They have been there before, so they know what they are talking about.

3.3.4.3 Estimation; Delphi Technique

This is a simple technique that has proved remarkably resilient even in highly complex situations.

You must appoint an estimation group as appropriate. This can be stakeholders and/or experts in the tasks to estimate.

The steps in this estimation process are:

- Each member of the group gives an estimate.
- The group is informed about the average and distribution of the estimates.
- Those giving estimates in the lower quartile and in the upper quartile are asked to tell the rest of the group why their estimates were as they were.
- The group estimates again—this time taking the previous result and the provided arguments for the "extreme" estimates into account.
- This may continue two, three, four, or more times until the variation in the estimates is sufficiently small.

Usually the average of the estimations does not change much, but the variation is rapidly decreased. This gives confidence in the final estimation result.

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 110 2/19/08 8:14:16 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

The Delphi techniques can be used in many ways. The people taking part can be in the same room, but they may also be continents apart and the technique used via e-mail.

The technique can be combined with other techniques. Most often the participants give their initial estimates based on experience and/or they are experts in a specific area. The initial estimates may also be obtained using some of the other estimation techniques to make them even more trustworthy.

3.3.4.4 Estimation; Three-Point Estimation

Three-point estimation is a statistical calculation of the probability of finishing within a given time. The technique is useful for quantifying uncertainty to the estimate. The technique is also called successive calculation because tasks are broken down and the estimates successively calculated until the variance is within acceptable limits.

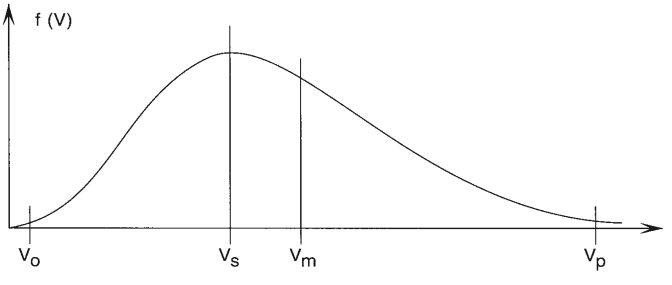
Three point estimation is based on three estimates:

- The most optimistic time (ideal conditions)
- The most likely time (if we do business as usual)
- The most pessimistic time (Murphy is with us all the way)

The three estimates to be used can be provided in a number of ways. One person can be asked to provide all of them, or a group, for example some of the test team members, can take part in the estimation. The estimates can be provided using the Delphi technique or other recognized techniques. High and low values may either be estimated separately (i.e., "what are the best and the worst cases?") or they may be the highest and the lowest of the individual estimates.

If the "best and worst and most likely" values are used, the estimators should be 99% sure that the actual value will fall between the low and the high values.

From these three estimates it is possible to define the distribution function for the time to finish. It could look like the figure shown where V_o = most optimistic; V_s = most likely; V_p = most pessimistic; and V_m = mean.



Copyright © 2008. Artech House. All rights reserved.

Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Book-sample.indb 111 2/19/08 8:14:16 PM

We can use the approximated formula to derive:
$V_m = (V_o + 3*V_s + V_p) / 5$
$S = (V_p - V_o) / 5$ (the standard deviation)
Based on this distribution we can calculate the time needed for any probability of finishing the task we want, by using the appropriate formula.

For the 5% interval the formulas are:
5%: Vm – 2S 95%: Vm + 2S
Let us say that for a testing task we have reckoned:
Vo = 70 hours Vs = 80 hours Vp = 110 hours
We calculate:
Vm = (Vo + 3*Vs + Vp)/5 = (70 + 3*80 + 110)/5 = 84
S = (Vp – Vo)/5 = (110 – 70)/5 = 8
The upper value in the 95% interval = 84 + 2 * 8 = 100
Therefore if we want to be 95% sure that we'll finish in time our estimate for the given task should be 100 hours.

All tasks or a selection of the most critical tasks can be estimated using this technique.

3.3.4.5 Estimation; Function Points

This technique is a factor estimation technique initially published by Albrecht in 1979. It has been revised several times, and it now maintained by IFPUG —International Function Points User Group. The group has been permanent since 1992. Version 4.0 of the technique was published in 1994.

The estimation is based on a model of the product, for example, a requirements specification and/or a prototype.

Five aspects of the product are counted from the model:

- External inputs
- External outputs
- External enquiries
- Internal logical files
- External interface files

The counts are then multiplied with a weight and the total of the weighted counts is the unadjusted sum. The actual effort in person hours is then calculated with an adjustment factor obtained from previous project data.

It requires some training to be able to count function points correctly. Continuous comparisons of actual time spent with the estimates are essential to get the best possible local adjustment factor.

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 112 2/19/08 8:14:16 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

The disadvantage of using function points is that they require detailed requirements in advance. Many modern systems are specified using use cases, and use cases are incompatible with this technique.

3.3.4.6 Estimation; Test Points

In 1999 Martin Pol et al. published a dedicated test estimation technique called test points as part of the TMAP method.

The technique is based on the function point technique, and it provides a unit of measurement for the size of the high-level test (system and acceptance tests) to be executed.

The technique converts function points into test points based on the impact of specific factors that affect tests, such as:

- Quality requirements
- The system's size and complexity
- The quality of the test basis (the document(s) the test is specified toward)
- The extent to which test tools are used

3.3.4.7 Estimation; Percentage Distribution

Unlike all the other techniques discussed here, this technique is a so-called top-down estimation technique. The fundamental idea is that test efforts can be derived from the development effort.

The estimation using this technique starts from an estimate of the total effort for a project. This estimate may be the result of the usage of appropriate estimation techniques at the project management level.

The next step is to use formulas (usually just percentages) to distribute this total effort over defined tasks, including the testing tasks. The formulas are based on empirical data, and they vary widely from organization to organization.

It is essential that you get your own empirical data and constantly trim it according to experiences gained.

If you do not have any data you could assume that the total testing effort is 25–30% of the total project effort. The testing effort should then be spread out on the test levels with an appropriate amount for each level.

This example is from Capers Jones *Applied software measurements*. It is for in-house development of administrative systems. The left-hand table shows the distribution of the total effort on overall tasks, including all tests as one task only. The right-hand table shows the distribution of the effort on detailed testing tasks (the terminology is that of Capers Jones.)

Copyright © 2008. Artech House. All rights reserved.

Book: samfer.indb 113 2/19/08 8:14:17 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Activity	%
Requirements	9.5
Design	15.5
Coding	20
Test (all test phases)	27
Project management	13
Quality assurance	0
Configuration management	3
Documentation	9
Installation and training	3

All phases	%
Component testing	16
Independent testing	84
	100
Independent testing	%
Integration testing	24
System testing	52
Acceptance testing	24
	100
System testing	%
Functional system testing	65
Nonfunctional system testing	35
	100

3.3.5 From Estimations to Plan and Back Again

The estimation is done to provide input to the scheduling activity in the project planning.

In the scheduling we bring the estimates for the defined testing tasks together with the people, who are going to be performing the tasks. Based on the start date for the first task and the dependencies between the tasks we can then puzzle the tasks together and calculate the expected finishing date.

Estimations should be in hours. The scheduling provides the dates: dates for when the performance of each of the tasks should begin, and dates for when they are expected to be finished.

When defining the expected finish date for a task we need to take several aspects into account:

- The start and/or finish dates of others tasks that this task depends on to start, if any
- The earliest possible start date for the task
- The general calendar regarding public holidays
- The pure estimate for the time to finish the task
- The efficiency of the employee(s) to perform the task—typically 70–80% for a full time assignment
- The employee(s)'s availability—this should NOT be less than 25%

We should not expect that our estimations are accepted straightaway. Making plans for a development project is a very delicate balance between

Copyright © 2008. Artech House. All rights reserved.

Book: seiffert.indb 114 2/19/08 8:14:17 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebook Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

resources (including cost), time, and quality of the work to be done. Testing is often on the critical path for a project, and testing estimates are likely to be the subject of negotiations between stakeholders—typically the customer or higher management, the project manager, and the test manager.

The estimating does not stop with the preparation of the first schedule. Once the actual testing has started—from the first planning activities and onwards, we need to frequently monitor how realities correspond to the estimates. Based on the new information gathered through the monitoring, we must re-estimate, when the deviations between estimates and reality get too large to stay in control. Only when all the testing activities are completed can we stop the monitoring and re-estimation.

3.3.6 Get Your Own Measurements

All estimates are based on experience—maybe very informally (FIA), maybe very formally (like function points). The better the basis for the estimation is, the better the estimation gets. Better estimation means more reliable estimations, and that is what we both, management and customers, want.

In order to get better estimates we need to collect actual data. The more empirical data we have, the better will the estimates be. In general we can say that (almost) any data is better than no data.

We do, however, always need to objectively evaluate the empirical data we have—is it collected from tasks that can be compared with the ones we are dealing with now? When we use the empirical data available, we also have an obligation to contribute to and refine the empirical data on an ongoing basis.

Empirical data for estimation is part of the measurements we are collecting. So we need to chip in to establish a set of simple measurements of time, costs, and size in all projects we participate in. This requires procedure(s) for collection of data to be established and maintained, and training of the (test) managers in these procedure(s).

From an organizational point of view it must be checked that all completed projects collect data, and the usage of these data must, of course, also be enforced.

3.4 Test Progress Monitoring and Control

Continuous monitoring of how the test is progressing compared to the plan is absolutely necessary to stay in control. If we don't control the test project, it will control us—and that is not a nice experience.

You need to collect information about facts, compare these with the estimates, and analyze the findings. This is needed to minimize divergence from the test plan. If there is any discrepancy you need to take action to keep in control, and you need to inform the stakeholders.

Copyright © 2008. Artech House. All rights reserved.

Book: samfer.indb 115 2/19/08 8:14:17 PM
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

There are a few rules that you must adhere to when you do the follow-up on the actual activities. Follow-up must be guided by:

- Honesty
- Visibility
- Action

First of all you need to be honest, not only when you estimate, but also when you collect information about reality. In the long run you loose integrity and trust if you "tailor" the numbers, or come up with "political" results of the monitoring.

You also need to make the information visible to all stakeholders. Again you lose trust if you hide the truth, be it a positive truth (we are ahead of schedule) or a negative truth (we are behind schedule). Information about progress and findings must be made readily available to the stakeholders in appropriate forms.

The last thing you need to do to stay in control is to take action whenever needed. *It is your duty as test manager to intervene as soon as deviations appear!*

3.4.1 Collecting Data

The data to collect during testing should be specified in the approach section in the test plan, based on the requirements outlined in the policy and the strategy.

The concept of metrics and measurements is discussed in Section 1.3.

No matter which data we have planned to collect it is not enough to just collect it. It must be presented and analyzed to be of real value.

3.4.2 Presenting the Measurements

Test reports are used to communicate test progress. These reports must be tailored to the different recipients or stakeholders. The immediate stakeholders for test monitoring information are the customer, the project and/or product management (or higher), the test management, and the testers.

The customer and the management above test management need test reports (described below) when the test is completed. The test management needs information on a continuous basis to keep in control. The testers need to be kept informed on progress at a very regular basis—at least daily when the activities are at their peak.

A picture speaks a thousand words. The best way to present progress information for testing is by using graphics. This holds true for all stakeholders, though especially for the testers in the middle of the action. Graphics used in the right way give an immediate overview—or feeling—for the state of the testing.

Copyright © 2008. Artech House. All rights reserved.

Book-sample.indb 116 2/19/08 8:14:18 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

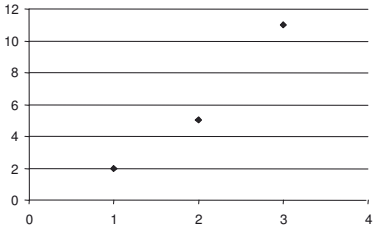
Sources:
Tufle and Huff

The flip side of the coin is that graphics can "lie." You can do it deliberately —which is outside the scope of this book—or you can make it happen accidentally if you are too eager to make your presentation "interesting" and "lively." The truth is usually boring, but adding decoration does not help.

One of the common mistakes is to use too many dimensions. Most of our information is one-dimensional: the number of something. Many graphs, however, present one-dimensional information in a two- or even three-dimensional way.

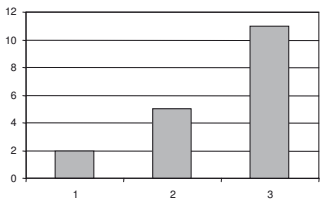
Consider the following information:

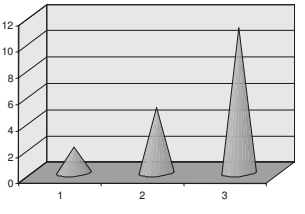
Day 1:	2 faults found
Day 2:	5 faults found
Day 3	11 faults found



The simplest way to present this is as shown to the right. See the trend? Yes, that is perfectly clear! Need anything else? Not really.

But all too often we may see exactly the same information presented like this: or, even worse, like this:





Does that add to the understanding? No.

There is a "metric" called the ink-factor. That is defined as the amount of ink used to convey the message compared to the amount of ink used in the graph. You should keep the ink-factor as low as possible.

Also *avoid highlighting (read: hiding) the message in decoration, patterns, shading, or color*. A graph that presents the number of failures found each day as the size of the corollas of a line of flowers is perhaps cute, but not professional.

More obvious ways to misinform is by changing the scale across the axis, or by omitting or distorting the context of the information or the way it has been collected.

Copyright © 2008. Artech House. All rights reserved.

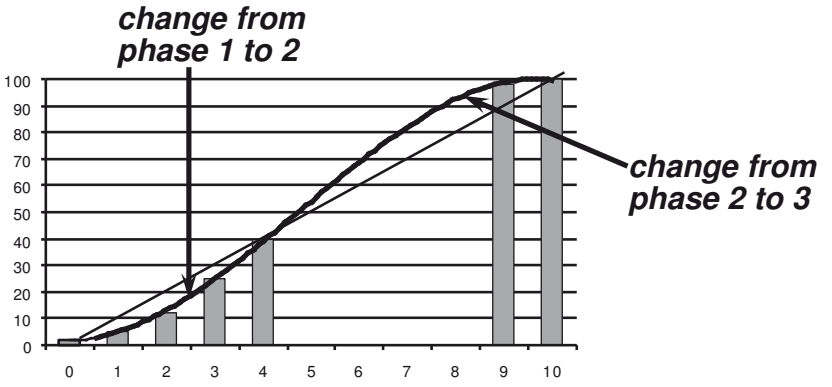
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Book: samlet.indb 117 2/19/08 8:14:18 PM

Whichever way you choose to present the information you have collected, it is your responsibility to ensure that the recipients understand and interpret the data correctly. In the following some of the most common and useful ways of presenting test progress information are described.

3.4.2.1 S-Curves

The most used, most loved, and most useful way of presenting progress information and controlling what's happening is S-curves. They are named so because of the shape of the wanted curve.



Source: Marnie Hutcheson, Unicom Seminar, Oct 95.

S-curves can be used for many different metrics. You need two that are related to each other—one is typically time. The other could, for example, be:

- Test cases (run, attempted, passed, completed)
- Incidents (encountered, fixed, retested)

S-curves can give us early warnings of something being wrong. It can also give us reassurance that (so far) things are progressing as planned.

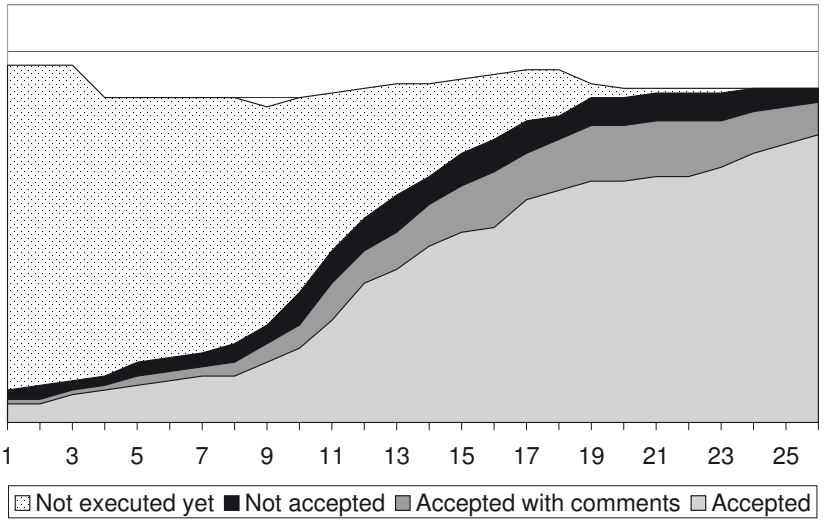
The principle in S-curves is that our work falls in three phases:

- Phase 1: Slow start—not more than 15–25%
  An initial period of reduced efficiency to allow for testing teams to become familiar with the testing task, for example with the test object, the test environment, and execution and logging practices.
- Phase 2: Productive phase—55–65%
  After the initial period, the second phase is the period of maximum efficiency.
- Phase 3: The difficult part—10–25%
  The final phase reflects the need to be able to ease off the work as the testing window nears completion.

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 118 2/19/08 8:14:19 PM
Mette, Jonassen Haug, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

The following figure shows how real data are reported as several S-curves in the same graph.



To use an S-curve you need to know what the expected start point (usually 0,0) and the expected end point are. The end point is your estimation of what needs to be achieved by the end time; for example 300 test cases passed after 21 days of testing.

You mark the start point and the end point, and you draw (or get a tool to draw) a third-order polynomial that fits. A straight-line approximation between the two points may do.

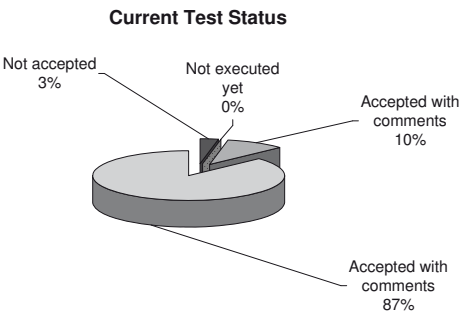
As the time goes and you do you work, you plot in your achievements—for example, the sum of test cases passed day by day. Follow the progress to see if it fits the predicted curve. If it does, we are happy!

If the upward turn, marking the start of the second phase, comes too late, we are in trouble. But the good news is that we know it and can take action well before the end date! If the curve is rising too fast, we may also be in trouble, and we must investigate what could be wrong. Maybe our test cases are not giving enough failures? Maybe we have run the smallest first and are pushing work in front of us?

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 119 2/19/08 8:14:19 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

3.4.2.2 Pie Chart



Pie charts are used to give an overview of the proportions of different aspects relative to each other. Pie charts are perhaps the most used graph in the world.

The graph shown here gives a nice impression of the testing going well.

But think about the inkfactor—maybe the third dimension is not needed to present the information. Maybe it is even disturbing the impression: Should the lightest gray volume be almost nine times as big as the medium gray?

3.4.2.3 Check Sheets

Check sheets are a good way to give a quick overview of status. They can be used to show progress compared to plan, for example, for planned test cases, planned test areas, or defined risks.

Check sheets can be presented as colorful graphics or expressed as lists or hierarchies. They are usually easy to produce, and easy to understand. Some organizations make wall-sized check sheets and stick them in the meeting room or the corridor. This way everyone has easy access to the information about the progress of the test.

A few examples of check sheets are shown next.

The first is an extract of the check sheet presented on Systematic's intranet. It is updated every day. Even though the text has been deliberately blurred and the extract is small, it gives an impression of things going well.—no black or light gray fields in the list!

Ex.

Status for project: ksdf				
Area	**Remain**	**% Complete**	**Status**	**Comment**
agfdg	8	56		
gstyk	0	100		
jl,flli	0	100		
dsrahjtdulk	1	80		
ths	2	56		
jdvw	0	100		
yjdtek	0	100		
	0	87		

Legend
- Completed
- In progress
- Blocked (see comment)
- Not started

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 120 2/19/08 8:14:20 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Ex.

The next is a dashboard suggested by James Bach:

Area	Test effort	Coverage planned	Coverage achieved	Quality	Comments
Start up	High	>80%	27%	😐	ER 52
Discount	Low	>40% <70%	53%	☺	
Pricing	**Blocked**	>40% <70%	14%	☹	**ER 86**

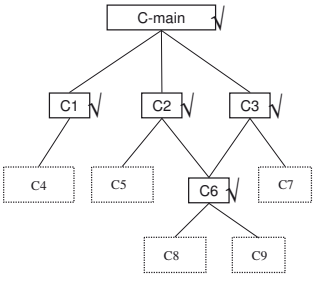
James Bach's recommendations for the presentation are: draw it on the wall, make it huge, and update it every day.

Ex.

The last example here is a tiny extract of a hierarchical check sheet showing the progress of a component test for a system.

The marked components have been successfully component-tested and are ready for integration.

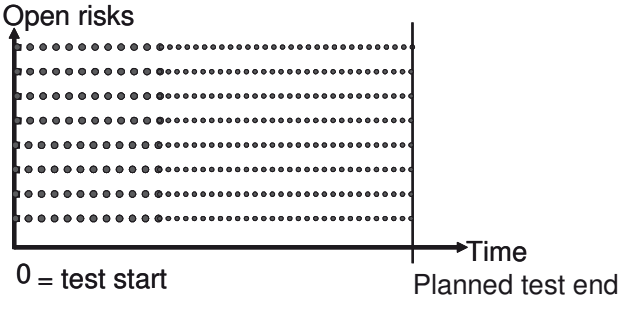
The integration testing has not yet started—no interfaces are marked as having been successfully tested. It is, however, easy to see which interfaces we have to test.



3.4.2.4 Risk-Based Reporting

If our test approach is based on identified and analyzed risks it is appropriate to report on the test progress in terms of these risks.

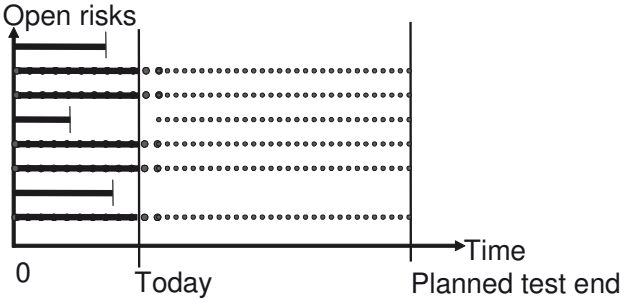
The purpose of this test is to eliminate the risk, so the reporting must be in terms of eliminated risks. The open risks at the test start could be illustrated like this:



Copyright © 2008. Artech House. All rights reserved.

Book1.sample.indb 121 2/19/08 8:14:22 PM
Mette, Jonassen Haug, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebook Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

At any point in time we must make it possible to see from the updated progress graph which risks are still open, if any. The risks with the small vertical line across them are eliminated and hence no longer present any threat to the system!



3.4.2.5 Statistical Reporting

The way a process is performed is different from project to project and over time, because processes are performed by people, not machines.

Statistics is the science of patterns in a variable world. We can say that statistics make the invisible visible. This means that statistical methods can be used to help us:

- Understand the past
- Control the present
- Predict the future

Statistics also include handling of "fortuitousness," that is, happenings that are out of the ordinary.

When we have to deal with many happenings assumed to be "alike," we need to find out what "alike" means. To do that we must find out what the norm is, and what variances are allowed to still call things "normal."

Ex.

Norm and variation vary. In our family the norm is that we are friendly and talk to each other in nice, calm tones of voices. I, however, have a short temper, and sometimes raise my voice without anything being really out of the normal. If, on the other hand, I keep quiet, then my mood is not within the norm. My husband is different: If he raises his voice just a little bit, he is sure to be in a very bad mood.

The norm in statistics can be calculated in three much used ways, namely: mean—the arithmetic average; median—the value that splits the group in the

Copyright © 2008. Artech House. All rights reserved.

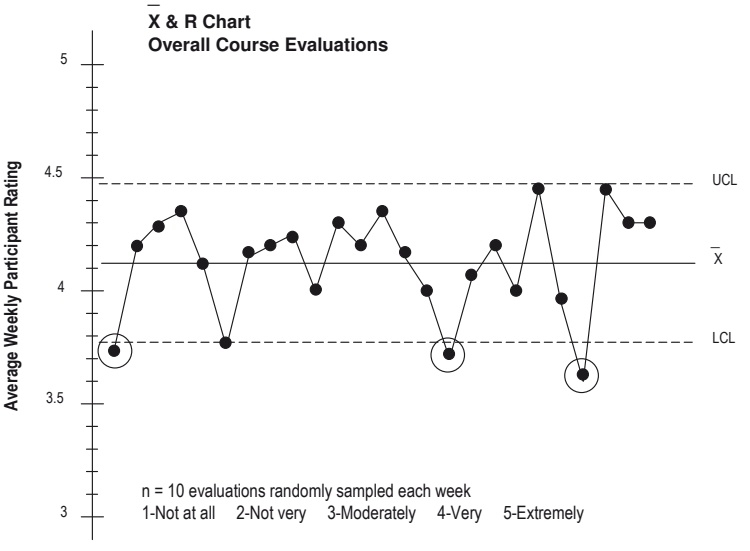
Book: samfet.indb 122 2/19/08 8:14:23 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

middle; and modus—the most frequent value.

But how far from the "normal" value, can a given value be and still be considered within the norm? This can be seen in a control sheet.

Ex.

An example of a control sheet is shown here. The values are ratings for a course. Every week 10 evaluations are randomly sampled and the average is plotted in the graphs. The graph shows the upper and lower control levels (UCL and LCL) for the series of ratings.



The values here are indicators for the course performance. We can choose other values to be indicator values for our processes, if we want to control how they are performed.

Ex.

An indication value may, for example, be the average time per test case it takes to produce a test specification.

When we examine the control sheet we must be looking for warnings of something being out of the borders of the norm. Such warnings may, for example, be:

- One value outside either CL
- Two out of three values on the same side
- Six values in a row either up or down
- Fourteen values in a row alternately up and down

Copyright © 2008. Artech House. All rights reserved.

Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Book: sample.indb 123 2/19/08 8:14:23 PM

Many tools can assist in the necessary statistical calculations. The use of control sheets and statistics is rather advanced process control—belonging to maturity level 4—and we will not go into further depth here.

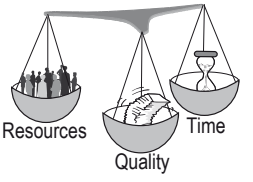
3.4.3 Stay in Control

Sometimes we need to take action to stay in control.

Keeping the triangle of test quality in mind, you have three aspects you can change—and you must change at least two at the time.

The aspects are:

- The resources for the task
- The time for the task
- The quality of the work to be performed



Usually when things are getting out of control it is because we are behind schedule or because our time frame has been squeezed. To compensate for this we must (try to) obtain additional resources and/or change the quality of the testing. The latter can be done by changing the test completion criteria and/or changing the amount or depths of the tests to be performed.

Any change you make must be reflected in the plan. The plan must be updated with the new decisions based on the new information. The new plan must be reviewed and approved, just like the first one.

Remember that *it is not a virtue to comply with the plan at any cost—the virtue lies in the plan complying with reality.*

No matter which way the progress is measured and presented, the test manager must analyze the measurements. If something seems to be going the adverse way, further analysis must be made to determine what may be wrong.

Examples of things not being as they should be are:

- The delivered software is not ready for test
- The easy test cases have been run first
- The test cases are not sufficiently specified
- The test case does not give the right coverage
- General faults have wide effects
- Fault correction is too slow
- Fault correction is not sufficiently effective

Based on this analysis the test manager must identify what can be done to remedy or mitigate the problems.

Copyright © 2008. Artech House. All rights reserved.

Book.sam[et.indb 124 2/19/08 8:14:24 PM
Mette Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Possible actions may, for example, be:

- Tighten entry criteria
- Cancel the project
- Improve the test specifications
- Improve the incident reporting
- Perform a new risk analysis and replan
- Do more regression testing

The important message to the test manager is that he or she must intervene as soon as deviations appear! Or in other words:

If you do not control the test, it will control you!

To sum up we can say that as test managers we must set out the destination and plan how to get there; collect data as we go along; analyze data to obtain information; and act on the information and change destination and plan as appropriate.

3.5 Testing and Risk

The golden rule of testing is:

Always test so that whenever you have to stop you have done the best possible test.

Everybody with some understanding of requirements and tests will know that a requirement like this cannot be verified. What does it mean: the best possible test? It is not immediately measurable.

What is the best possible test then? The answer to that is: It depends!

The best possible test depends on the risk associated with having defects left in the product when it is released to the customer!

The best possible test is determined by the risks we are facing and the risks we are willing to run. Obtaining consensus from stakeholders on the most important risks to cover is essential.

3.5.1 Introduction to Risk-Based Testing

We have to live with the fact that it is impossible to test everything. Testing is sample control. There is a risk involved in all sample control: the risk of overlooking defects in the areas we are not testing.

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 125
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:24 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

3.5.1.1 Risk Definition

A risk is defined as: "The possibility of realizing an unwanted negative consequence of an incident."

Alternatively, a risk may be defined as: *"A problem that has not materialized yet and possibly never will."*

There are two important points in these definitions:

- A risk entails something negative
- A risk may or may not happen—we don't know

A risk therefore has two aspects:

- Effect (impact—consequence)
- Probability (likelihood—frequency)

It is not a risk (to us), if there is no effect of an event that might happen. We can therefore ignore it even if the probability is high.

There is a probability that there are defects in the new version of our database management system, but that will have no effect on the quality of our product if it happens, because it is not used in our system.

It is not a risk if there is no (or an extremely small) probability that an event will happen, even if the effect would be extremely big, if it did. We can therefore ignore that as well.

There is no (detectable) risk of our department closing down, because we have lots of orders, and are making good money, and both management and employees like their jobs. If we did close down the effect on the project would be pretty bad, if not disastrous.

It is not a risk either if the probability of an event with a negative effect is 100%. In this case we have a real problem on our hands, and we will have to deal with that in our planning.

It is a problem—not a risk—that we will have to do without one of our test experts because she has found another job and is leaving in three months.

The two aspects of risk can be combined in

Risk level = probability x effect

From this it is quite clear that if we have no probability or no effect we have no risk.

The risks that do have a risk exposure greater than zero are the risks we have to deal with.

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 126 2/19/08 8:14:24 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

3.5.1.2 Risk Types

It is quite common to treat all the risks we can think of in connection with a development project in one big bundle. This can be quite overwhelming.

It is therefore a very good idea to take a closer look at the risks and divide them into classes, corresponding to where they may hit, or what they are threatening.

Risks hit in different places, namely;

- The business
- The processes
- The project
- The product

The risks threatening the product are the testers' main concern. This is where we can make a difference.

The business risks are things threatening the entire company or organization from a "staying-in-business" point of view. This is out of the scope of this book, and will not be discussed further.

Process risks are related to the processes and/or the way work is performed. It is also out of the scope of this book, but will be briefly discussed because knowledge about processes is indispensable in a modern development organization.

Process risk threatens the effectiveness and efficiency with which we work on an assignment. Process risks may be originated in:

- Missing process(es)
- The organization's lack of knowledge about the processes
- Inadequate processes
- Inconsistencies between processes
- Unsuitable processes
- Lack of support in the form of templates and techniques

Process risks jeopardize the way the work in the project is being performed. These risks should be the concern of the project manager and those responsible for the processes in the organization.

Process risks may influence business, as well as project and product risks.

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 127
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:25 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

Project Risks

Project risks are related to the project and the successful completion of the project.

A project consists of a number of activities and phases from requirements development to the final acceptance test. These activities are supported by activities like quality assurance, configuration management, and project management.

All the activities in a project are estimated, get allocated resources, and are scheduled. As the project progresses the activities are monitored according to the plan.

Risks concerning the project may be originated in:

- People assigned to the project (e.g., their availability, adequate skills and knowledge, and personalities)
- Time
- Money
- Development and test environment, including tools
- External interfaces
- Customer/supplier relationships

Project risks jeopardize the project's progress and successful completion according to the plan.

Examples of project risks are:

- The necessary analysts are not available when the requirements development is expected to start
- Two of the senior designers are not on speaking terms and useful information exchange between them is not done—this causes the design phase to take longer than expected
- The complexity of the user interface has been underestimated
- The testers are not adequately trained in testing techniques, so testing requires more resources than expected
- The integration is more time-consuming than expected
- The access to external data is not possible with the technique chosen in the design

The project risks are the main concern of the project manager and higher management.

The test manager is concerned with the project risks related to the test project as it is specified in the test plan documents.

Copyright © 2008. Artech House. All rights reserved.

Book1.samp.indb 128 2/19/08 8:14:25 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Product Risks

Product risks are related to the final product. They are the risks of defects remaining in the product when it is delivered.

We want to deliver the required quality and reliability. This cannot be tested into the product at the end of the development, but must be worked into the product through the work products produced during development and in the implementation of the components.

Product risks may be originated in:

- Functional and nonfunctional requirements
- Missing requirements
- Ambiguous requirements
- Misunderstood requirements
- Requirements on which stakeholders do not agree

More than 50% of all defects in products can be traced back to defects in the requirements.

- Design
- Not traceable to requirement
- Incorrect
- Incomplete (too superficial)
- Coding
- Testing
- Not traceable
- Inadequate

Product risks jeopardize customer satisfaction and maybe even the customer's life and livelihood.

Product risks may be related to different requirement types, such as functionality, safety, and security and political and technical factors.

Examples of product risks are:

- A small, but important functionality has been overlooked in the requirements and is therefore not implemented
- A calculation of discounts is wrongly implemented, and the customer may lose a lot of money
- The instrument may reset to default values if it is dropped on the floor
- It is possible to print a report of confidential customer information through a loophole in the reporting facility
- The installation procedure is difficult to follow

Copyright © 2008. Artech House. All rights reserved.

Book1.sample.indb 129 2/19/08 8:14:25 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

These risks are the main concern of the testers, since testing may mitigate the risks. The test strategy for a product should be based on the product risks that have been identified and analyzed.

Project risks and product risks can influence and be the cause of each other. A project risk may cause a product risk, and a product risk may cause a project risk.

If a project risk results in time being cut from component testing, this may cause the product risk of defects remaining in the components that are not tested or not tested sufficiently. This may further cause the project risk that there is not sufficient time to perform a proper system test because too many trivial failures are encountered in the system test.

3.5.1.3 Testing and Risk Management

Testing and management of risks should be tightly interwoven as they support each other. Testing can be based on the results of risk analysis, and test results can give valuable feedback to support continuous risk analysis.

The result of a product risk analysis can be used in test planning to make the test as effective as possible. It can be used to target the testing effort, since the different types of testing are most effective for different risks.

Component testing is most effective for testing where the product risk exposure related to complex calculations is highest.

The risk analysis results can also be used to prioritize and distribute the test effort. The areas with the risk exposure should be planned to be tested first and given the most time and other resources.

Finally the product risk analysis can be used to qualify the testing already done. If test effort is related to risks it should be possible to report on dissolved and remaining risks at any time.

Testing can, as mentioned earlier, dissolve or mitigate the product risks. The probability of sending a product with defects out to the customers is reduced by the testing finding failures and the subsequent correction of the defects.

Testing can also mitigate project risks if an appropriate test strategy is applied, especially if testing is started early.

Even process risks may be reduced by analyzing failure reports and taking appropriate process improvement initiatives.

Copyright © 2008. Artech House. All rights reserved.

Book1.sample.indb 130 2/19/08 8:14:25 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

3.5.2 Risk Management

Risk management consists of the following activities:

- Risk identification
- Risk analysis
- Risk mitigation
- Risk follow-up

In risk identification we are finding out what may happen to threaten the process, the project, or, in this particular context, the customer satisfaction for the product.

The identified risks are evaluated in the risk analysis and ordered relatively to each other. The analysis means that we assign probability and effect to the risks. Based on this we can determine the risk exposure and hence which risk is the worst and how the others relate to that.

One of the points in risk management is to use the results of the analysis to mitigate the risks. Actions can be planned to lower the probability and/or the effect of the risks. In this context of product risks and testing, test activities can be planned to mitigate the risks by lowering the probability of having remaining defects in the product when it is released. The more defects we can remove from the product as a result of the testing, the more the probability falls.

Contingency planning is a part of classic risk management, but this is not relevant for product risks in relation to testing. Testing is concerned with lowering the probability of remaining defects for the defects that remain; support and maintenance must be prepared to provide work-arounds and/or corrections and updates.

Risk identification, analysis, and mitigation must not be a one-time activity. It is necessary to follow-up on the risks as testing progresses. The results of the testing activities provide input to continuous risk management.

Information about the failure frequency over time can be used to assess if the probability of a risk is falling or rising.

There are many stakeholders in a development project, and they come from different places and have different view points, also on risks.

The following table shows examples of stakeholders or stakeholder representatives for a number of aspects of a development project.

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 131 2/19/08 8:14:26 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Aspect	Stakeholder representatives
The business	Product line manager Business analysts Marketing personnel Finance executives
Future users	Future users Users of existing system Users representing different types of user groups
User representations	Marketing personnel Salespeople Employees from relevant support function
Technical, development	Analysts, designers, programmers, testers People responsible for manufacturing People responsible for operation People responsible for configuration management People responsible for quality assurance
Technical, product	Experts in and people responsible for usability, security reliability, performance, portability

All stakeholders identified for a project should be involved in the risk management activities.

3.5.2.1 Risk Identification

Risk identification is finding out where things can go wrong and what can go wrong, and writing it down to form the basis for risk analysis.

Risks are found in areas where the fulfillment of expectations may be threatened (i.e., where customer satisfaction is jeopardized). *Satisfying expectations is the way to success!*

All stakeholders have expectations towards the product, but we are most concerned with the expectations of the customer. The customer is the one to order the product, to pay for it, and to take advantage of it, the latter possibly through end users.

In the ideal world the customer's expectations are expressed in user or product requirements. These requirements are transformed into design, and the requirements are fulfilled in the code and maybe also in other subsystems, being integrated into the final products.

In less ideal worlds expectations may be derived from other sources.

Risks are not always evident. Even when we work with experienced and knowledgeable stakeholders it can be efficient to use a risk identification technique.

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 132 2/19/08 8:14:27 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Useful techniques are:

- Lessons learned
- Checklists
- Risk workshops
- Brainstorms
- Expert interviews
- Independent risk assessments

Techniques may be mixed to be even more efficient.

3.5.2.2 Lessons Learned and Checklists

Lessons learned and checklists are closely related. A checklist is a list of generic risks formed and maintained by experience (i.e., lessons learned from previous projects).

Risk checklists are valuable assets in an organization and should always be treated as such.

One or more product risk checklists should be kept in the organization, depending on the diversity of the nature of the projects performed in the organization.

A product risk checklist could be structured as the requirements specification or the design specification is structured, or both.

The checklists used by the pilots before takeoff are long and must be run through very carefully before every takeoff. A pilot was once hurried on by a busy business man and asked to drop the checklists and get going. He carried on with his work as he answered: "These checklists are written in blood!"

3.5.2.3 Risk Workshops

Workshops are an effective way to identify risks. There are no strict rules as to how a workshop should be conducted, but a few guidelines can be given.

As many stakeholders as possible should be involved, though the number of participants should not exceed 10–12 in order to give everybody a chance to talk within reasonable intervals.

Risk workshops can get emotional, and a neutral facilitator—somebody who is not by any account a stakeholder—should be present to guide the discussions. *Encourage discussions and new ideas, but avoid conflicts.*

Make sure that all participants agree that the objective has been reached and that it is clear how work can proceed after the workshop.

Copyright © 2008. Artech House. All rights reserved.

Book: samfen.indb 133
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:27 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

3.5.2.4 Brainstorming

A brainstorm (in this context of risk identification) is an informal session with the purpose of identifying possible risks connected with the product when it is released.

The only rule that should apply during a brainstorm *is that no possible risk must be commented on* in any way by the participants. Ideas should be allowed to flow freely, the rationale being that even the most seemingly stupid, silly, or strange thought may be the inspiration for valuable potential risks.

A brainstorm must have a facilitator who may act as a catalyst if ideas do not flow freely. At the end of the session the facilitator must make sure that whatever is being brought forward as possible risks is expressed as risk and documented.

The stakeholders to be involved in this technique may be any type of stakeholders.

3.5.2.5 Expert Interviews

Interviews may be conducted as individual interviews or as group interviews. *An interview is not as easy to conduct* as many people think. It requires specific skills and thorough preparation to get as much information as possible from an interview.

First of all, an interview is not like an ordinary conversation. People in an interview have different roles (i.e., the interviewer and the interviewee(s)), and they may have a number of expectations and prejudices related to these roles. Interviews must be prepared. The interviewer must, for example, make sure that the right people are being interviewed and the right information is gathered. A list of questions or a framework for the course of the interview must be prepared.

Ample notes must be taken and/or the interview can be recorded with the permission of the interviewee(s). *The interviewer must extract a list of possible risks from the interview* and get agreement from all the participants.

3.5.2.6 Independent Risk Assessments

In cases where conflicts are threatening external consultants may be called in to identify risks. External consultants could also be used if time is short or if specific expertise not present with the immediate stakeholders is required.

The external consultants identify risks and usually perform or facilitate the risk analysis.

The consultants may be external to the project organization or third-party consultants entirely external to the developing organization.

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 134 2/19/08 8:14:27 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

3.5.3 Risk Analysis

Risk analysis is the study of the identified risks. One thing is to identify and list the risks; another is to put them into perspective relative to each other. This is what the analysis of the risks helps us do.

The analysis must be performed by all the appropriate stakeholders, because they all have different perspectives and the risk analysis aims at providing a common and agreed perspective. Experts may be called in to contribute if adequate expertise cannot be found among the immediate stakeholders.

Risk analysis can be performed more or less rigorously, but it should always be taken seriously.

3.5.3.1 Risk Template

A risk template or a risk register is a very useful tool in risk management. It can be used to support risk analysis and risk mitigation.

Risk templates can be held in office tools, for example, spreadsheets that support calculations.

A risk template should include:

- Risk identification (e.g., number or title)
- Risk description
- Probability
- Effect
- Exposure
- Test priority
- Mitigation action = test type
- Dependencies and assumptions

3.5.3.2 Perception of Risks

The performance of risk analysis can be more or less objective. In fact *most risk analysis is based on perceptions*; it is usually not possible to determine risk probability and effect totally objectively. There is an element of prediction in risk analysis since we have to do with something that has not happened and maybe never will. There are usually very few trusted measurements applicable to identified risks.

Perceptions are personal and different people have different "pain thresholds." Just look around you: Some people use their holidays to explore new places; others always spend their holidays at the same place. In connection with process improvements we sometimes say that if it blows hard some people build shelters; others build windmills.

"If you do that, I'll never see you again!!!"
"Is that a promise or a threat?"

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 135 2/19/08 8:14:27 PM
Mette, Jonassen Haug, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebook Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

People in different professions may also have different view points on risks, partly because people choose jobs according to their personalities, partly because job-related experiences influence their perception of different risks.

The following descriptions of job-related risk perceptions are of course gross generalizations, but they can be used as guidelines in understanding different viewpoints on "the same risks." The descriptions encompass:

- Project managers
- Developers
- Testers
- End users

Project managers are often under time pressure; they are used to compromises. They know that even though things may look dark, the world usually keeps standing.

Developers, that is analysts, designers, and programmers, are proud of their work, and they know how it was done. They have really done their best, and they are usually reluctant to accept that there may still be defects left in there.

Testers often have a pessimistic view on work products, product components, and products. We remember previous experiences where we received objects for testing and got far more failures than we expected.

The *end users* are, despite what we might think, usually highly failure-tolerant. They also tend to remember previous experiences, but what they remember is that even though the system failed, they found a way around it or another way of doing their work. End users use our product as a tool in their job, and nothing more. If it does not help them, they'll find another way of using the tool, find another tool, or just live with it.

In risk analysis we must encourage communication and understanding between stakeholders. *Stakeholders need to be able to, if not agree with then at least be aware of and accept others' points of view.* If need be, stakeholders will have to compromise or use composite analysis. This is explained next.

3.5.3.3 Scales for Risk Analysis

The analysis of risks uses metrics for probability and effect. For all work with metrics it is mandatory to use agreed and understood scales. This is therefore also the case in risk analysis.

We can work with two different kinds of scale, namely:

- Qualitative
- Quantitative

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 136 2/19/08 8:14:28 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

In a qualitative scale we work with feelings or assessments.

For effect we could use
bad—worse—worst
For probability this could be expressed as
not likely—likely—very likely

In a quantitative scale, on the other hand, we work with exact measures or numbers.

For effect we could use actual cost in $ or Kr. or €.
For probability this could be expressed as
<= 10%, >10% & <= 50%, >50% & <=80%, > 80%

Whichever way to do it, we must define and agree on scales for both probability and effect before we start the risk analysis, that is before we assign metrics to the probability of the identified risks actually materializing, and metrics for the effects if they do.

3.5.3.4 Effect

The effect is the impact or consequences of a risk if (when?) it occurs. The first thing we have to do is agree on a scale for the effect.

The obvious *quantitative scale* for the effect is the actual cost imparted by a failure occurring out in the field. The actual cost can be measured in any agreed currency ($, €, Kr.). This is an open scale; in theory there is no limit to actual cost.

It can be very difficult to assess what the actual cost in real money might be. On the other hand it can be quite an eye-opener to sit down and consider all the sources of extra cost associated with a failure.

Expenses may for example be considered for:

- Time for the end user to realize that something is wrong
- Time to report the incident to first-line support
- Time for first-line support to understand the report and try to help
- Time for any double or extra work to be performed by the end users
- Loss of production because the system is down or malfunctioning
- Time for escalation to secondhand support
- Time for secondhand support to try to help
- Time to investigate the failure and decide what to do about it
- Time for finding the defect(s)
- Time for corrections to be implemented and tested in all affected objects

Copyright © 2008. Artech House. All rights reserved.

Book1.sam1e.indb 137 2/19/08 8:14:28 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

- Time for retesting and regression testing
- Time to reinstall the new version
- Time to update what has been done by other means while the system was unavailable or malfunctioning

These are all examples of time spent in connection with a failure. There may also be costs associated with, for example, renting or replacing parts of the system or the entire system.

Furthermore there may be an effect in the form of indirect losses from, for example, people getting hurt, the environment being destroyed, or the company getting an adverse reputation or losing trustworthiness.

Failures have been known to cost lives or to put companies out of the market completely. Fortunately, it is usually not that bad, but still the effects of failures can be significant.

Another way to measure effect is by using a qualitative scale. Such a scale could, for example, be expressed as shown in the following table.

Inspired by Paul Gerrard

Effect	Description	Score
Critical	Goals cannot be achieved	6
High	Goals will be jeopardized	5
Above middle	Goals will be significantly affected	4
Below middle	Goals will be affected	3
Low	Goals will be slightly affected	2
Negligible	Goals will be barely noticeably affected	1

In the table there is a column for a mnemonic for the effect, a column describing the effect more precisely, and a column for the actual score.

Using a numeric score makes it possible to calculate the risk exposure even when a qualitative scale is used for the effect.

Despite the above example it can be useful to define a scale with an even number of scores. This can mitigate the effect of some people having a tendency for choosing the middle value if they are not sure what to score or can't be bothered to think deeper about their opinion. A scale with an even number of scores does not have a middle value and the stakeholders will have to decide if they want to score over the middle or under.

The important point before the analysis of the effects can start is that *the stakeholders agree to and understand the scale.*

When you perform the analysis of the effect of the risks you have identified, you must keep your focus on the effect. You must *NEVER let the probability influence the effect!* It can sometimes be tempting to give the effect an extra little turn upwards if we know (or think) that the probability of the risk materializing is high. This will give a twisted picture of the risk exposure and should be avoided.

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 138 2/19/08 8:14:28 PM
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

A simple effect analysis for the risks pertaining to the four top-level architectural areas defined for a product may look like this, using a scale from 1–6 where 6 is worst.

Risk area	Effect
Setup	2
Conveyor	2
Concentration calculation	6
Compound determination	5

Often it is not enough to have one single score for the effect. Stakeholders see the effect from different perspectives. An end user sees the effect in the light of how a failure will influence his or her daily work. A customer may look at the effect of failures on the overall business goals. A supplier organization may assess the effect in terms of correct efforts for failures or loss of credibility in the market.

These different perspectives can be honored if we use a more complex or composite effect analysis. The score should be the same for all the perspectives, but the descriptions should be tailored to make sense for each of the viewpoints.

A composite effect analysis taking more perspectives into account may look like this:

Risk area	Effect for perspective			Final effect
	User	Customer	Supplier	
Setup	5	3	2	3.3
Conveyor	3	3	5	3.7
Concentration calculation	2	5	2	3
Compound determination	1	5	3	3
Here all perspectives have the same weight, and the final effect is a simple average of the effect contributions.				

If the scale is not sufficiently differentiated the individual perspectives may be assigned independent weights, and the final effect can then be calculated as the weighted average:

$$\text{Final effect} = \sum(\text{effect}*\text{weight}) / \sum(\text{weight})$$

The effect analysis taking more perspectives into account and assigning different weights to the perspectives may look as shown next.

Copyright © 2008. Artech House. All rights reserved.

Book: samfen.indb 139
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.
2/19/08 8:14:29 PM

Ex.

Risk area	Effect for perspective			Final effect
	User W=2	Customer W=7	Supplier W=1	
Setup	5	3	2	3.3
Conveyor	3	3	5	3.2
Concentration calculation	2	5	2	4.1
Compound determination	1	5	3	3.9

3.5.3.5 Probability

The probability is the likelihood of the materialization of a risk.

Also here we first of all need to agree on a scale. On a *quantitative scale* probability can be measured on a scale from 0 to 1 or a scale from 0% to 100%. For most risks it is, however, almost impossible to determine the probability with such a precision.

A *qualitative scale* for probability is usually much more useful, as long as it doesn't get too loose, like "likely," "very likely," "extremely likely."

A qualitative scale could be expressed as in the following table where there is a column for probability intervals, a column describing the probability, and a column for the actual score. Again using a numeric score makes it possible to calculate the risk exposure even when a qualitative scale is used for the probability.

Ex.

Probability	Description	Score
99–82	Highly likely	6
81–	Likely	5
–50	Above 50–50	4
49–	Below 50–50	3
Low	Unlikely	2
–1	Highly unlikely	1

Inspired by Paul Gerrard

For effect, you must keep your focus on the probability when you perform the analysis of the probability of the risks you have identified. You must *NEVER let the effect influence the probability!* It can sometimes be tempting to give the probability an extra little turn upwards if we know (or think) that the effect of the risk if it materializes is high. This will give an untrue picture of the risk exposure and should be avoided.

The probability of a risk materializing may be a function of many factors, for example:

Copyright © 2008. Artech House. All rights reserved.

Book: seiler.indb 140 2/19/08 8:14:29 PM
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

- Complexity of the product or the code
- Size of the product or the code
- The producer of the work product(s) or component(s)
- Whether it is a new product or code or maintenance
- The previous defect record for the product or area
- The developers' familiarity with tools and processes

Just like it is explained for the effect above the final probability can be calculated as the weighted average of the probabilities pertaining to the different factors.

Final probability = ∑(probability*weight) / ∑(weight)

A composite probability analysis may look like this:

Risk area	Probability for factor			Final prob.
	Size W=1	History W=5	Complexity W=2	
Setup	4	2	1	2
Conveyor	5	3	5	3.8
Concentration calculation	3	1	2	1.5
Compound determination	3	1	5	2.2

The same quantitative scale must be used for all the factors.

3.5.3.6 Risk Level

The risk level is calculated for each of the identified risks as

Risk level = final effect x final probability

It is not a difficult task to perform a risk analysis as explained above. A full analysis including identifying about 30 risks and assessing and calculating the effect, probability, and final level can be done in a couple of hours. It is well worth the effort because it gives everybody involved a much clearer picture of why test is necessary and how the test should be planned.

Spreadsheets can be used for easy calculation of the levels and for maintenance of the risk analysis.

Using the above examples for final effect and final probability, the final risk level may look as shown in the following table.

Copyright © 2008. Artech House. All rights reserved.

Book1.samp.indb 141 2/19/08 8:14:30 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebook Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Risk area	Final effect	Final probability	Final risk level
Setup	3.3	2	**6.6**
Conveyor	3.3	3.8	**12.2**
Concentration calculation	4.1	1.5	**6.2**
Compound determination	3.9	2.2	**8.6**

It sometimes happens that some *stakeholders are unhappy with the final level.* If a stakeholder has high rates for a particular risk and the risk comes out with a relatively low final risk level, this can "seem unfair." In such cases the perspectives and the scales will have to be discussed once more.

The point of the perspectives and the scales is that they should satisfy every stakeholder's viewpoint. If that is not the case they must be adjusted. Most of the time, however, stakeholders recognize that the perspectives and scales are OK and that their viewpoint is fairly overruled by others, different viewpoints.

The distribution of the final risk level over individual risks is used to plan the test activities. It can be used to prioritize the test activities and to distribute the available time and other resources according to the relative risk level. A test plan based on a risk analysis is more trusted than a plan based on "gut feeling."

It is difficult to predict events, and therefore all risk analysis has some built-in uncertainty. *A risk analysis must be repeated at regular intervals as the testing progresses.*

The testing results can be used as input to the continuous risk analysis. If we get more defects than expected in a particular area it means that the probability is higher than we expected, and the area hence has a higher risk level. On the other hand if we get fewer defects than expected the risk level is lower.

3.5.4 Risk Mitigation

We use the results of the risk analysis as the basis for the risk mitigation, the last activity of the sequential risk management activities. "To mitigate" means "to make or become milder, less severe or less painful." That is what we'll try to do.

Faced with the list of risks and their individual risk level we have to go through each of the risks and decide:

- What we are going to do
- How we are going to do it
- When we are going to do it

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 142 2/19/08 8:14:30 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

3.5.4.1 What to Do to Mitigate Risks

In terms of what to do we have the following choices:

- Do nothing
- Share the pain
- Plan contingency action(s)
- Take preventive action

We can choose to do nothing if the benefit of waiting to see how things develop is greater than the cost of doing something.

You would not buy a safe for € 1,000 to protect your jewels if they were only worth € 500 (including the sentimental value). If the jewels were stolen you could buy new ones and still have money left.

Sharing the pain is outside the scope of testing, but it is a possibility for the project management or higher management to negotiate sharing the pain of the effect of a materializing risk with other parties. This other party could be an insurance company or it could be a supplier or even the customer.

Planning contingency action is a natural part of most risk mitigation. The contingency action is what we are going to do to mitigate the effect of a risk once it actually has materialized. For other risk types than product risks and other processes than testing the response to the risk analysis may be production of contingency plans. But it is not something that is applicable in the test planning for mitigating product risks.

Extra courses are planned if it turns out that the system is more difficult to learn than expected.

Testing is one of a number of possible preventive actions. The aim is to mitigate the risks. Testing can be used to mitigate the risk exposure by lowering or eliminating the probability of the risk.

Product risks are associated with presence of defects. The effect is associated with the effect if a defect causes a failure of the product in use. The probability is associated with the probability of undetected defects still being present in the product when it is released.

Testing aims at identifying defects by making the product fail—before it reaches the customer. Defects found in testing can be corrected—before the system reaches the customer. Hence testing and defect correction reduces the risk level by reducing the probability.

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 143
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:30 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

3.5.4.2 How to Mitigate Risks by Testing

When we have decided to do something to mitigate a risk, we must find out what to do. The nature of the risk can be used to determine the phases and types of testing to perform to mitigate the risk and the level of formality applied. The decisions must be documented in the applicable test strategy or test plan.

Certain test phases are especially applicable for certain types of risks. We need to look at the risk source and determine the phases and activities that are most likely to unveil defects.

Some examples are given next.

Risk source	Recommended test phases
High risk of defects in algorithms	◗ Review of detailed design ◗ Review of code ◗ Component testing ◗ Functional system test
Risk of problems in the user interface	◗ Usability evaluation of prototype (requirements review) ◗ Usability test (nonfunctional system test)
Risk of performance problems	◗ Performance test (nonfunctional system test)
Risk concerning external interface	◗ Review of design ◗ Review of code ◗ System integration test

The formality of the test can also be determined from the risk exposure. The rule is simple:

The higher the risk level => The higher the formality

The formality can change from level to level and it can change over the product. Some areas can have more formal testing than others, even within the same test level.

At any level, for example, system testing, we can have the different levels for formality as shown in the following table.

Copyright © 2008. Artech House. All rights reserved.

Book1.sample.indb 144 2/19/08 8:14:31 PM
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

System test

Ex.

Risk level	Recommended test phases
High	◗ Specific test case design techniques to be used ◗ Strict test completion criteria ◗ Strict regression test procedures
Low	◗ Free choice of test case design techniques ◗ Less strict test completion criteria ◗ Less strict regression test procedures

3.5.4.3 When to Mitigate Risks by Testing

We can use the results of the risk analysis to prioritize the test activities that we have identified for the risks and to distribute the test time (and possibly other resources).

In the prioritization we are going to determine the order in which to attack the risks. Even if we are not going to perform all the testing activities identified for the risks in strictly sequential order, it is a help in the planning to have them prioritized.

The priority can follow the final risk exposure. This means that the final exposure can be as the sorting criteria. This takes every perspective of the risks into consideration in one attempt.

With the example from above the priority of the risk areas can then be as shown in the following, where 1 is the highest.

Ex.

Risk area	Final risk level	Priority
Setup	**6.6**	3
Conveyor	**12.2**	1
Concentration calculation	**6.2**	4
Compound determination	**8.6**	2

The stakeholders could also choose to let the prioritization be guided by either the final effect or the final probability, or they may even agree to use one particular perspective, for example, the probability related to the complexity, to prioritize from.

In order to calculate the distribution of the time to spend on the testing we need to calculate the sum of the final exposure.

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 145
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:31 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

Ex.

Risk area	Final level
Setup	6.6
Conveyor	12.2
Concentration calculation	6.2
Compound determination	8.6
Total	33.6

The next step is to calculate the distribution of the final levels over the risk areas. This could look as shown here, where the percentages have been rounded to the nearest whole number.

Ex.

Risk area	Final level	% distribution
Setup	6.6	20 %
Conveyor	12.2	36 %
Concentration calculation	6.2	10 %
Compound determination	8.6	26 %
Total	33.6	100 %

With a table like this we have a strong planning tool. No matter which resource we have at our disposal we can distribute it on the risk areas and hence ensure that each area is indeed tested, but neither more nor less than it deserves.

If the project manager for example allocates 400 hours for the complete test of our example system, we can distribute this time over the areas as shown here:

Ex.

Risk area	% distribution	Hours for testing
Setup	20 %	80
Conveyor	36 %	144
Concentration calculation	18 %	72
Compound determination	26 %	104
Total	**100**	**400**

The list of prioritized risks with their allocated resources and identified testing phases and activities allows us to produce a substantiated plan and schedule for the test.

The list also allows us to immediately assess the results of a proposed change in resource allocation. If the resource we have distributed is cut, we will have to find out how to make do with what is left.

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 146 2/19/08 8:14:32 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebook Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Usually we are operating with time; having a number of hours allocated for the testing and consequently a number of hours may be cut. If time is cut we must ask management what to do with our distribution of time on the risks. We can't leave our plan and schedule untouched; the cut must have an effect. What we can do is, in principle:

- Reduce testing time proportionally to the cut
- Take risk areas out of the testing completely

The best thing to do is to reduce the time proportionally. This ensures that all areas are still tested, that is, we will get some information relating to all the risks. We can combine the two approaches but we should be very careful if we take areas out completely.

If some testing has already been performed, a renewed risk analysis is necessary before we can act on any cuts. In this case we must distribute the remaining resources over the remaining risks according the new final exposure and prioritize as we did before or by a new, more relevant criterion.

Questions

1. How does testing provide business value?
2. What is the purpose of testing?
3. What is product reliability?
4. On what does the cost of defect correction depend?
5. Where do most of the defects originate from?
6. What does good testing provide?
7. What four types of test management documents are defined?
8. What is the purpose of the test policy?
9. What should the test policy include?
10. What could a quality target for example be?
11. What is the purpose of the test strategy?
12. What are the approaches to the testing?
13. What are the strategy topics to be considered?
14. Why may standards be useful?
15. What should a strategy include in relation to risks?
16. Why are completion criteria important?
17. What is the idea in degree of independence?
18. What may be reused in testing?
19. Why does the strategy need to be specific about tools?
20. Why should we measure during testing?
21. To which support process does the incident management belong?
22. What is a master plan for?

Copyright © 2008. Artech House. All rights reserved.

Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Book: sample.indb 147 2/19/08 8:14:32 PM

23. What is a level test plan?
24. What are the 14 topics that should be covered in a test plan according to IEEE 829?
25. What should the introduction contain?
26. What is the relationship between the test item and features to be tested?
27. Why is it important to describe what is not tested?
28. What should the approach description consider?
29. For what are pass/fail criteria used?
30. What may cause a pause in the test execution?
31. What are the typical test deliverables?
32. What is the important thing concerning testing tasks?
33. How can you illustrate who carries which responsibility?
34. In which part of the test plan do the testing tasks, the estimates, and the people come together?
35. Who should approve the test plan?
36. What does it mean that a (test) plan must be SMART?
37. What is *The New Yorker* way of planning?
38. Who may be interested in test reports?
39. What topics should be covered in a test summary report according to IEEE 829?
40. What is a comprehensiveness assessment?
41. Who must decide if a test object should be released?
42. What is estimation?
43. What is not estimation?
44. How is test estimation different from ordinary project estimation?
45. What are the six steps in the estimation process?
46. What are the three estimation technique types?
47. On what is the analogy technique based?
48. What are the steps in the Delphi technique?
49. What three estimates do we need for the successive calculation estimation technique?
50. What is the estimation based on in the successive calculation estimation technique?
51. What are the five things you count for function point calculation?
52. What is the test estimation technique based on function points?
53. What is the difference between all the other techniques and the percentage distribution technique?
54. What must be taken into account when defining the finish date for a task?
55. When can we stop estimating?

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 148
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:32 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

56. What must guide our follow-up activities?
57. What is the virtue in planning?
58. On what should we base our follow-up?
59. How can we present measurements?
60. What is the ink-factor?
61. What is the principle in S-curves?
62. What is a check sheet?
63. What is risk-based reporting?
64. Why should we use statistical reporting for process performance?
65. What must we do to stay in control?
66. What can we do?
67. What is the golden rule of testing?
68. What is a risk?
69. What is risk exposure?
70. What is a process risk?
71. What is a project risk?
72. What could a project risk be?
73. What is a product risk?
74. How does testing relate to risks?
75. What are the activities in risk management?
76. Which stakeholders could be involved in the work with risks?
77. What is a risk template?
78. What could a product risk be?
79. Which risk identification techniques could we use?
80. For what is a risk checklist used?
81. What is the main rule for brainstorms?
82. How must an expert interview be prepared?
83. What is the output from a risk interview?
84. What is risk analysis?
85. How may the viewpoints of risk differ for different roles?
86. What kinds of scales can be used for risk analysis?
87. What must not be done when assessing effect?
88. What is a composite effect analysis?
89. How do you calculate the final effect?
90. What must not be done when assessing probability?
91. Which factors could be used in composite probability assessment?
92. For what is the result of a risk analysis used?
93. How many times should a risk analysis be performed, and why?
94. What is mitigation?
95. What are the actions that can be taken as a result of a risk analysis?
96. Which action is most appropriate for product risks?

Copyright © 2008. Artech House. All rights reserved.

Book: samfer.indb 149
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:32 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

97. Which test activities and test levels could be used to mitigate different risks?
98. How are the formality of testing and risk exposures related?
99. How can you prioritize the testing for risks?
100. How can you use risk exposure to distribute testing resources?
101. What can you do if testing time is cut?

Copyright © 2008. Artech House. All rights reserved.

Book.samp.indb 150 2/19/08 8:14:32 PM
Mette, Jonassen Haas. Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

CHAPTER

4

Test Techniques

Contents

Test case design techniques are the heart of testing. There are many advantages of using techniques to design test cases. They support systematic and meticulous work and make the testing specification effective and efficient; they are also extremely good for finding possible faults. Techniques are the synthesis of "best practice"—not necessarily scientifically based, but based on many testers' experiences.

Other advantages are that the design of the test cases may be repeated by others, and that it is possible to explain how test cases have been designed using techniques. This makes the test cases much more trustworthy than test cases just "picked out of the air." The test case design techniques are based on models of the system, typically in the form of requirements or design. We can therefore calculate the coverage we are obtaining for the various test design techniques.

Coverage is one of the most important ways to express what is required from our testing activities. It is worth noticing that coverage is always expressed in terms related to a specific test design technique. Having achieved a high coverage using one technique only says something about the testing with that technique, not the complete testing possible for a given object.

Test case design techniques have a few pitfalls that we need to be aware of. Even if we could obtain 100% coverage of what we set out to cover (be it requirements, or statements, or paths), faults could remain after testing simply because the code does not properly reflect what the users and customers want. Validation of the requirements before we start the dynamic testing can mitigate this risk.

There is also a pitfall in relation to value sensitivity. Even if we use an input value that gives us the coverage we want it may be a value for which incidental correctness applies. An example of this is the fact that:

2 + 2 equals 2 * 2; but 3 + 3 does not equal 3 * 3!

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 151 2/19/08 8:14:33 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

4.1 Specification-Based Techniques

The specification-based case design techniques are used to design test cases based on an analysis of the description of the product without reference to its internal workings. These techniques are also known as black-box tests.

These techniques focus on the functionality. They are dependent on descriptions of the expectations towards the product or system. These should be in the form of requirements specifications, but may also be in the form of, for example, user manuals and/or process descriptions. If we are lucky we get the requirements expressed in ways corresponding directly to these techniques; if not we'll have to help analysts do that during requirements documentation or do it ourselves during test design.

These test case design techniques can be used in all stages and levels of testing. The techniques can be used as a starting point in low-level tests, component testing and integration testing, where test cases can be designed based on the design and/or the requirements. These test cases can be supplied with structural or white-box tests to obtain adequate coverage.

The techniques are also very useful in high-level tests like acceptance testing and system testing, where the test cases are designed from the requirements.

The specification-based techniques have associated coverage measures, and the application of these techniques refines the coverage from requirements coverage to specific coverage items for the techniques.

The functional test case design techniques covered here are:

- Equivalence partitioning and boundary value analysis
- (Domain analysis—not part of the ISTQB syllabus)
- Decision tables
- Cause-effect graph
- State transition testing
- Classification tree method
- Pairwise testing
- Use case testing
- (Syntax testing—not part of the ISTQB syllabus)

4.1.1 Equivalence Partitioning and Boundary Value Analysis

Designing test cases is about finding the input to cover something we want to test. If we consider the number of different inputs that we can give to a product we can have anything from very few to a huge amount of possibilities.

Copyright © 2008. Artech House. All rights reserved.

Book.sample.indb 152 2/19/08 8:14:34 PM
Mette Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

A product may have only one button and it can be either on or off = 2 possibilities.

A field must be filled in with the name of a valid postal district = thousands of possibilities.

It can be very difficult to figure out which input to choose for our test cases. The equivalence partitioning test technique can help us handle situations with many input possibilities.

4.1.1.1 Equivalence Partitioning

The basic idea is that we partition the input or output domain into equivalence classes.

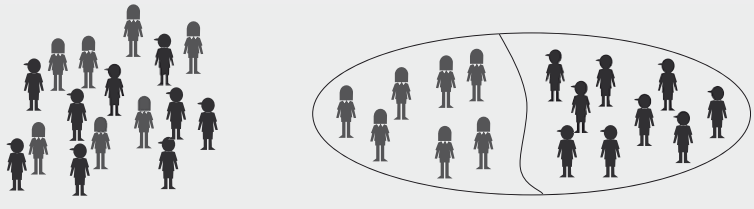
A class is a portion of the domain. The domain is said to be partitioned into classes if all members of the domain belong to exactly one class—no member belongs to more than one class and no member falls outside the classes.

The term equivalence refers to the assumption that all the members in a class behave in the same way. In this context the assumption is based on the requirements or other specifications of the product's expected behavior.

The reason for the equivalence partitioning is that all members in an equivalence class will either fail or pass the same test. One member represents all! If we select one member of a class and use that for our test case, we can assume that we have tested all the members.

Choosing test cases based on equivalence partitioning insures representative test cases.

If we take a class of pupils and the requirement says that all the girls should have an e-mail every Thurdays reminding them to bring their swimsuits, we can partition the class into a partition of girls and a partition of boys and use one representative from each class in our test cases.



When we partition a domain into equivalence classes, we will usually get both valid and invalid classes. The invalid classes contain the members that the product should reject, or in other words members for which the product's behavior is unspecified. Test cases should be designed for both the valid and the invalid classes, though sometimes it is not possible to execute test cases based on the invalid equivalence classes.

Copyright © 2008. Artech House. All rights reserved.

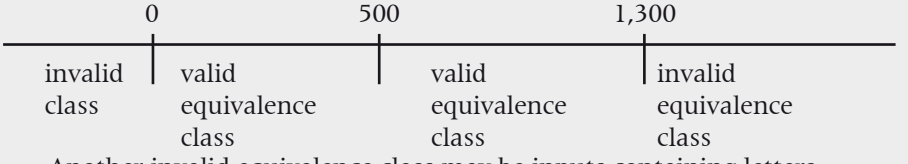
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

The most common types of equivalence class partitions are intervals and sets of possibilities (unordered list or ordered lists).

Intervals may be illustrated by a requirement stating:

Income in €	Tax percentage
Up to and including 500	0
More than 500, but less than 1,300	30
1,300 or more, but less than 5,000	40

If this is all we know, we have:



Another invalid equivalence class may be inputs containing letters.

To illustrate a set of possibilities we may use the unordered list of hair colors: (blond, brown, black, red, gray). Perhaps the product can suggest an appropriate dye for these colors of hair, but none other. The valid equivalence class is the list of values; all other values belong to the invalid class, assuming we don't know how the product is going to react, if we enter one such value.

It is possible to measure the coverage of equivalence partitions. The equivalence partition coverage is measured as the percentage of equivalence partitions that have been exercised by a test.

To exercise an equivalence class we need to pick one value in the equivalence class and make a test case for this. It is quite usual to pick a value near the middle of the equivalence class, if possible, but any value will do.

Test cases for the tax percentage could be based on the input values: –5; 234; 810; and 2,207.

For the hair colors we could choose: black and green, as a valid and on invalid input value, respectively.

4.1.1.2 Boundary Value Analysis

A boundary value is the value on a boundary of an equivalence class. Boundary value analysis is hence strongly related to equivalence class partitioning. Equivalence classes of intervals have boundaries, but those of lists do not. Boundary value analysis is the process of identifying the boundary values.

Copyright © 2008. Artech House. All rights reserved.

Book1.sam1et.indb 154 2/19/08 8:14:35 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

The boundary values require extra attention because defects are often found on or immediately around these. Choosing test cases based on boundary value analysis insures that the test cases are effective.

For interval classes with precise boundaries, it is not difficult to identify the boundary values.

The interval given as: 0 <= income <= 500, is one equivalence class with the two boundaries 0 and 500.

If a class has an imprecise boundary (> or <) the boundary value is one increment inside the imprecise boundary.

If the above interval had been specified as: 0 <= income < 500, and the smallest increment is given as 1, we would have an equivalence class with the two boundaries 0 and 499.

The smallest increment should always be specified; otherwise we must ask or guess based on common or otherwise available information.

It is often not specified what the increment is when we are dealing with money. A reasonable guess is that if we operate in euros (€), then the smallest increment is € 0.01 or 1 cent. But beware! Tax people usually use € 1 as the smallest increment.

Sometimes we'll experience equivalence classes with open boundaries—classes where a boundary is not specified. This makes it difficult to identify a boundary value to test. In these cases we must first of all try to get the specification changed. If that is not possible we can look for information in other requirements, look for indirect or hidden boundaries, or omit the testing of the nonexisting boundary value.

Open boundaries may be seen in connection with people's incomes. If the above interval had to do with incomes we may argue that the lowest income is € 0. But what is the real upper boundary? There is no obvious upper boundary for people's income.

Because we can have both valid and invalid equivalence classes we can also have both valid and invalid boundary values, and both should be tested. Sometimes, however, the invalid ones can not be tested.

When we select boundary values for testing, we must select the boundary value and at least one value one unit inside the boundary in the equivalence class. This means that for each boundary we test two values.

In traditional testing it was also recommended to choose a value one unit

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 155
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebook Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.
2/19/08 8:14:35 PM

outside the equivalence class, hence testing three values for each boundary. Such a value is in fact a value on the border of the adjacent equivalence class, and some duplication could occur. But we can still choose to select three values; the choice between two or three values should be governed by a risk evaluation.

It is possible to measure the coverage of boundary values. The boundary value coverage is measured as the percentage of boundary values that have been exercised by a test.

Equivalence Partitioning and Boundary Value Analysis Test Design Template

The design of the test conditions based on equivalence class partitioning and boundary value analysis can be captured in a table such as the following one.

Test design item number:		**Traces:**	
Based on: Input/Output		**Assumptions:**	
Type	**Description**	**Tag**	**BT**

Table designed by Carsten Jørgensen

The fields in the table are:
Test design item number: Unique identifier of the test design item
Traces: References to the requirement(s) or other descriptions covered by this test design
Based on: Input/Output: Indication of which type of domain the design is based on
Assumptions: Here any assumption must be documented.
For each test condition we have the following fields:
Type: Must be one of

- VC—Valid class
- IC—Invalid class
- VB—Valid boundary value
- IB—Invalid boundary value

Remember that the invalid values should be rejected by the system.
Description: The specification of the test condition
Tag: Unique identification of the test condition
BT = Belongs to: Indicates the class a boundary value belongs to. This can be used to cross-check the boundary values.

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 156 2/19/08 8:14:35 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Equivalence Partitioning and Boundary Value Analysis Test Design Examples

Ex.

In this example we shall find test conditions and test cases for the testing of this user requirement.

[UR 631] The system shall allow shipments for which the price is less than or equal to € 100.

The first thing we'll do is fill in the header of the design table.

Test design item number: 11	**Traces: [UR 631]**
Based on: Input	**Assumptions:** The price cannot be negative The smallest increment is 1 cent

The next thing is identifying the valid class(es).

Type	**Description**	**Tag**	**BT**
VC	0 <= shipment price <= 100		

We then consider if there are any invalid classes. If we only have the single requirement given above, we can identify two obvious and two special invalid equivalence classes. The new rows are indicated in bold.

Type	**Description**	**Tag**	**BT**
IC	**shipment price < 0**		
VC	0 <= shipment price <= 100		
IC	**shipment price > 100**		
IC	**shipment price is empty**		
IC	**shipment price contains characters**		

Our boundary value analysis gives us two boundary values.

Copyright © 2008. Artech House. All rights reserved.

Book.samp.indb 157 2/19/08 8:14:36 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Type	Description	Tag	BT
IC	shipment price < 0		
IB	**shipment price = –0.01**		
VB	**shipment price = 0.00**		
VC	0 <= shipment price <= 100		
VB	**shipment price = 100.00**		
IC	shipment price > 100		
IB	**shipment price = 100.01**		
IC	shipment price is empty		
IC	shipment price contains characters		

This concludes the equivalence class partitioning and boundary value analysis for the first requirement. We will complete the table by adding tags and indicating to which classes the boundary values belong.

Test design item number: 11	**Traces: [UR 631]**
Based on: Input	**Assumptions:** The price cannot be negative The smallest increment is 1 cent

Type	Description	Tag	BT
IC	shipment price < 0	11-1	
IB	shipment price = –0.01	11-2	11-1
VB	shipment price = 0.00	11-3	11-4
VC	0 <= shipment price <= 100	11-4	
VB	shipment price = 100.00	11-5	11-4
IC	shipment price > 100	11-6	
IB	shipment price = 100.01	11-7	11-6
IC	shipment price is empty	11-8	
IC	shipment price contains characters	11-9	

We can now make low-level test cases. If we want 100% equivalence partition coverage and two value boundary value coverage for the requirement, assuming that invalid values are rejected, we get the following test cases:

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 158 2/19/08 8:14:36 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Tag	Test case	Input Price=	Expected output
11-1	TC1-1	–25.00	rejection
11-2	TC1-2	0.02	rejection
11-2	TC1-3	–0.01	rejection
11-3	TC1-4	0.00	OK
11-3	TC1-5	0.01	OK
11-4	TC1-6	47.00	OK
11-5	TC1-7	99.99	OK
11-5	TC1-8	100.00	OK
11-7	TC1-9	100.01	rejection
11-7	TC1-10	100.02	rejection
11-6	TC1-11	114.00	rejection
11-8	TC1-12	" "	rejection
11-9	TC1-13	"abcd.nn"	rejection

We could choose to omit some of the test cases, especially since we actually get five different test cases covering the same equivalence class.

Ex.

In the next example we will test the following requirement:

[UR 627] The system shall allow the packing type to be specified as either "Box" or "Wrapping paper."

The test design table looks like this after the analysis.

Test design item number: 15		Traces: [UR 627]	
Based on: Input		Assumptions:	
Type	**Description**	**Tag**	**BT**
VC	"Box" "Wrapping paper"	15-1	
IC	All other texts	15-2	

This type of equivalence class does not have boundaries.

The test cases could be:

Tag	Test case	Input packing type	Expected output
15-1	TC3-1	"Box"	OK
15-2	TC3-2	"Paper"	rejection

It is only necessary to test one of the valid packing types, because it is a member of an equivalence class.

Copyright © 2008. Artech House. All rights reserved.

Book1.sample.indb 159 2/19/08 8:14:37 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebook Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Equivalence Partitioning and Boundary Value Analysis Hints

The equivalence partitioning and boundary value analysis of requirements makes it possible for us to:

- Reduce the number of test cases, because we can argue that one value from each equivalence partition is enough.
- Find more faults because we concentrate our focus on the boundaries where the density of defects, according to all experience, is highest.

Sometimes the input or output domains we are testing do not have one-dimensional boundaries as assumed in the equivalence class partitioning and boundary value analysis as described here.

If it is not possible or feasible to partition our input or output domain in one dimension we have to use the technique called domain analysis instead.

4.1.2 Domain Analysis

Note: This technique is not part of the ISTQB syllabus, but included here because I find it interesting and occasionally useful.

In equivalence partitioning of intervals where the boundaries are given by simple numbers, we have one-dimensional partitions. The domain analysis test case design technique is used when our input partitions are multi-dimensional, that is when a border for an equivalence partition depends on combinations of aspects or variables. If two variables are involved we have a two-dimensional domain; if three are involved, we have a three-dimensional domain, and so on.

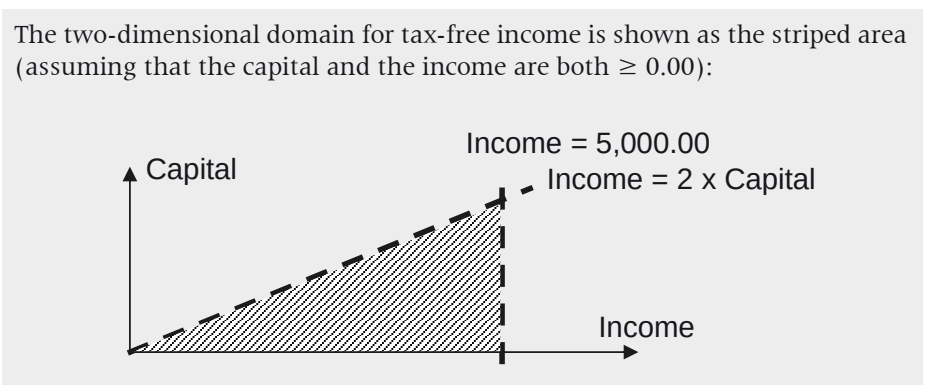
Multidimensional partitions are called domains—hence the name of the technique, even though the principles are the same as in equivalence partitioning and boundary value analysis.

It is difficult for people to picture more than three dimensions, but in theory there is no limit to the number of dimensions we may have to handle in domain analysis. We will use two-dimensional domains in the section; the principles are the same for any number of dimensions.

For equivalence partitioning we had the example of intervals of income groups, where 0.00 <= income < 5,000.00 is tax-free. This is a one-dimensional domain. If people's capital counts in the calculation as well, so that income is only tax-free if it is also less than twice the capital held by the person in question, we have a two-dimensional domain.

Copyright © 2008. Artech House. All rights reserved.

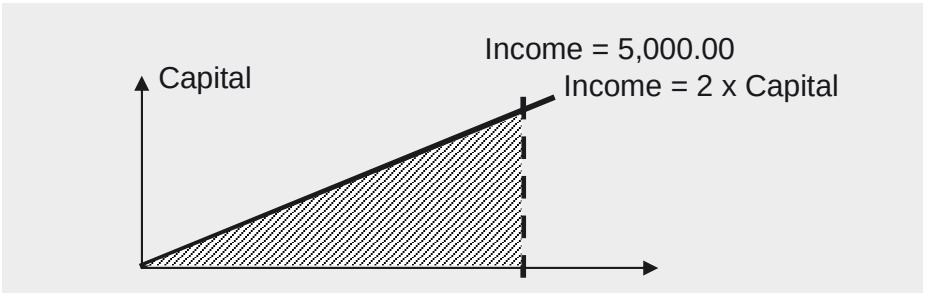
Book: sample.indb 160 2/19/08 8:14:37 PM
Meile, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.



Borders may be either open or closed. A border is *open*, if a value on the border does not belong to the domain we are looking at. This is the case in the example where both the borders are open (income < 5,000.00 and income < 2 x capital).

A border is *closed* if a value on the border belongs to the domain we are looking at.

If we change the border for tax-free income to become: income <= 2 x capital, we have a closed border. In this case our domain will look like this:



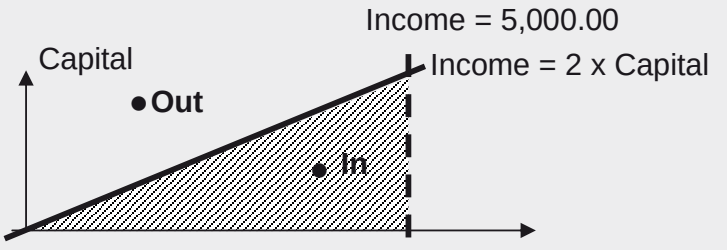
In equivalence partitioning we say that a value is in a particular equivalence class—in a way we do the same for domain analysis, though here we operate with points relative to the borders:

- A point is an In point in the domain we are considering, if it is inside and not on the border
- A point is an Out point to the domain we are considering, if it is outside and not on the border (it is then in another domain)

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 161
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:38 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

An In point and an Out point relative to the border income <= 2 x capital are illustrated here.

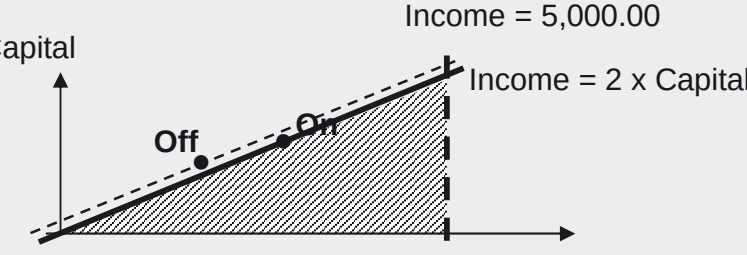


In the boundary value analysis related to equivalence partitioning described above, we operate with the boundary values on the boundary and one unit inside. In domain analysis we operate with On and Off points relative to each border.

We have:

- A point is an On point if it is on the border between partitions
- A point is an Off point if it is "slightly" off the border

If the border of the domain we are looking at is closed, the Off point will be outside the domain. The "slightly" may be one unit relative to what measure we are using, so that the Off point lies on the border of the adjacent domain. This works for all practical purposes, as long as we are not working with a floating point where "one unit" is undeterminable; in this case the "slightly" will have to be far enough away from the border to ensure that the Off point is inside the adjacent domain.

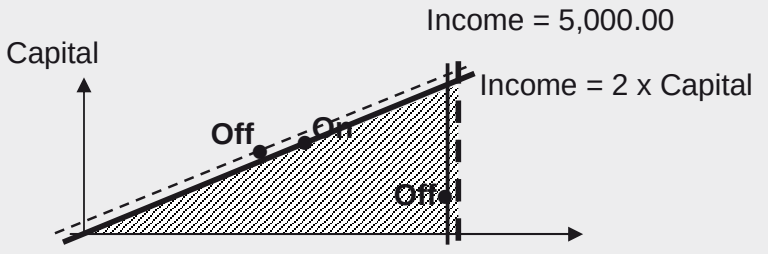


If the border of the domain we are looking at is open, the Off point will be "slightly" (or perhaps one unit) inside the domain, namely, outside (or on) the closed border of the adjacent domain.

Copyright © 2008. Artech House. All rights reserved.

Book1.sam1et.indb 162
Mette Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:38 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

In our case with the income < 5,000.00 an On point and an Off point may look like this:



Domain Analysis Strategy

The number of test cases we can design based on a domain analysis depends on the test strategy we decide to follow. A strategy can be described as:

N-On * N-Off

where N-On is the number of On points we want to test for each border and N-Off correspondingly is the number of Off points we want to test for each border for the domains we have identified.

If we choose a 1 * 1 strategy we therefore set out to test one On point and one Off point for each border of the domains we have identified. This is what is illustrated above for the one domain we are looking at (not taking the capital>=0 border into account). If our strategy is 2 * 1, we set out to test two On points and one Off point for each of the borders for all the domains we have identified. In this case the On points will be the points where the borders cross each other, that is the extremes of the borders.

In a 1 * 1 strategy we will get two test cases for each border. If we are testing adjacent domains, we will get equivalent test cases because an Off point in one domain is an In point in the adjacent domain. These test cases will have identical expected outcomes if identical values are chosen for the low-level test cases. These duplicates need not be repeated in the test procedures for execution.

Domain Analysis Coverage

It is possible to measure the coverage for domain analysis. The coverage elements for the identified domains are the In points and the Out points. The coverage is measured as the percentage of In points and Out points that have been exercised by a test. Do not count an In point in one partition being an Out point in another partition to be tested twice.

The coverage elements for the borders are the On points and Off points. The coverage is measured as the percentage of On points and Off points that have been exercised by a test relative to what the strategy determines as the number of points to test. Again do not count duplicate points twice.

Copyright © 2008. Artech House. All rights reserved.

Book: samfer.indb 163
Mette Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:39 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

Domain Analysis Test Design Template

The design of the test conditions based on domain analysis and with the aim of getting On and Off point coverage can be captured in a table like this one.

Tag						
Border 1 condition	ON	OFF				
Border 2 condition			ON	OFF		
Border n condition					ON	OFF

Table designed by Carsten Jørgensen

The table is for one domain; and it must be expanded both in the length and width to accommodate all the borders our domain may have.

The rule is: divide and conquer.

For each of the borders involved we should:

- Test an On point
- Test an Off point

If we want In point and Out point coverage as well, we must include this explicitly in the table.

When we start to make low-level test cases we add a row for each variable to select values for. In a two-dimensional domain we will have to select values for two variables.

Tag						
Border 1 condition	ON	OFF				
Border 2 condition			ON	OFF		
Border n condition					ON	OFF
Variable X						
Variable Y						

For each column we select a value that satisfies what we want. In the first column of values we must select a value for X and a value for Y that gives us a point On border 1. We should aim at getting In points for the other borders in the column, though that is not always possible. For each of the borders we should note which kind of point we get for the selected set of values.

This selection of values can be quite difficult, especially if we have high-dimensional domains. Making the table in a spreadsheet helps a lot, and other tools are also available to help.

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 164 2/19/08 8:14:39 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Domain Analysis Test Design Example

Ex.

In this example we shall test this user requirement:

[UR 637] The system shall allow posting of envelopes where the longest side (l) is longer than or equal to 12 centimeters, but not longer than 75 centimeters. The smallest side (w) must be longer than or equal to 1 centimeter. The length must be twice the width and must be greater than or equal to 10 centimeters. Measures are always rounded up to the nearest centimeter. All odd envelopes are to be handled by courier.

We can rewrite this requirement to read:
length >= 12
length < 75
width >= 1
length – 2 x width >= 10
This can be entered in our template:

Tag	TC1	TC2	TC3	TC4	TC5	TC6	TC7	TC8
Length > = 12	ON 12	OFF						
Length < 75	12 IN		ON	OFF				
Width > = 1	1 IN				ON	OFF		
1 – 2 x w > = 10	10 ON						ON	OFF
Length	12							
Width	1							

In the table we have entered values for the first test case, namely length = 12 to get the On point for the first border condition. The simultaneous selecting of width = 1 gives the points indicated for each border in lowercase under the number.

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 165
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:40 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

The table fully filled in may look like this:

Tag	TC1	TC2	TC3	TC4	TC5	TC6	TC7	TC8
Length > = 12	ON 12	OFF 11	75 in	74 in	15 in	15 in	30 in	31 in
Length < 75	12 IN	11 in	ON 75	OFF 74	15 in	15 in	30 in	31 in
Width > = 1	1 in	1 in	30 in	30 in	ON 1	OFF 0	10 in	11 in
1 – 2 x w > = 10	10 on	9 out	15 in	14 in	13 in	15 in	ON 10	OFF 9
Length	12	11	75	74	15	15	30	31
Width	1	1	30	30	1	0	10	11

We now need to determine the expected results, and then we have our test cases ready.

4.1.3 Decision Tables

A decision table is a table showing the actions of the system depending on certain combinations of input conditions.

Decision tables are often used to express rules and regulations for embedded systems and administrative systems. They seem to have gone a little out of fashion, and that is a shame. Decision tables are brilliant for overview and also for determining if the requirements are complete.

It is often seen that what could have been expressed in a decision table is attempted to be explained in text. The text may be several paragraphs or even pages long and reformatting of the text into decision tables will often reveal holes in the requirements.

Decision tables are useful to provide an overview of combinations of inputs and the resulting output. The combinations are derived from requirements, which are expressed as something that is either true or false. If we are lucky the requirements are expressed directly in decision tables where appropriate.

Decision tables always have 2^n columns, because there are always 2^n combinations, where n is the number of input conditions.

The number of rows in decision tables depends on the number of input conditions and the number of dependent actions. There is one row for each condition and one row for each action.

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 166
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:40 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

Input condition 1	T	T	F	F
Input condition 2	T	F	T	F
Action 1	T	T	F	F
Action 2	T	T	T	F

The table is read one column at a time. We can, for example, see that if both input conditions are true then both actions will happen (be true).

The *coverage measure* for decision tables is the percentage of the total number of combinations of input tested in a test.

Sometimes it is not possible to obtain 100% combination coverage because it is impossible to execute a test case for a combination.

4.1.3.1 Decision Table Templates

The template to capture decision table test conditions in is the template for the decision table itself with a test design header, as shown next.

Test design item number:	**Traces:**			
Assumptions:				
	TC1	TC2		TCn
Input condition 1				
Input condition n				
Action 1				
Action n				

The fields in the table are:

Test design item number: Unique identifier of the test design item
Traces: References to the requirement(s) or other descriptions covered by this test design
Assumptions: Here any assumption must be documented

The table must have a row for each input and each action, and 2^n columns, where n is the number of input conditions. The cells are filled in with either True or False to indicate if the input conditions, respectively the actions are true or false.

The easiest way to fill out a decision table is to fill in the input condition rows first. For the first input condition half of the cells are filled with True and the second half are filled with False. In the next row half of the cells under the Ts are filled with True and the other half with False, and likewise for the Fs. Keep on like this until the Ts and Fs alternate for each cell in the last input condition row.

The values for the resulting actions must be extracted from the requirements!

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 167
Mette Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:41 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

4.1.3.2 Decision Table Example

Ex.

In this example we are going to test the following requirements.

[76] The system shall only calculate discounts for members.

[77] The system shall calculate a discount of 5% if the value of the purchase is less than or equal to € 100. Otherwise the discount is 10%.

[78] The system shall write the discount % on the invoice.

[79] The system must write in the invoices to nonmembers that membership gives a discount.

Test design item number: 82	**Traces: Req. [76]–[79]**
Assumptions: The validity of the input is tested elsewhere	

Note that we are only going to test the calculation and printing on the invoice, not the correct calculation of the discount.

	TC1	TC2	TC3	TC4
Purchaser is member	T	T	F	F
Value <= € 100	T	F	T	F
No discount calculated	F	F	T	T
5% discount calculated	T	F	F	F
10% discount calculated	F	T	F	F
Member message on invoice	F	F	T	T
Discount % on invoice	T	T	F	F

The test cases to execute can now be created from these test conditions and written directly into a test procedure.

Test procedure: 11			
Purpose: This test procedure tests the calculation of discount for members. Traces: Req. [76]–[79]			
Tag	**TC**	**Input**	**Expected output**
TC1	1	Choose a member and create a purchase with a value less than € 100	A discount of 5% is calculated and this is written on the invoice.
TC2	2	Choose a member and create a purchase with a value of more than € 100	A discount of 10% is calculated and this is written on the invoice.
TC3	3	Choose a nonmember and create a purchase with a value less than € 100	No discount is calculated and the “membership gives discount” statement is written on the invoice.

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 168 2/19/08 8:14:41 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

4.1.3.3 Collapsed Decision Tables

Sometimes it seems evident in a decision table that some conditions are without effect because one decision is decisive. For example if one condition is False an action seems to be False no matter what the values of all the other conditions are.

This could lead us to collapse the decision table, that is reduce the number of combinations by only taking one of those where the rest will give the same result. This technique is related to the condition determination testing discussed below in Section 4.2.6.

The decision as to whether to collapse a decision table or not should be based on a risk analysis.

4.1.4 Cause-Effect Graph

A cause-effect graph is a graphical way of showing inputs or stimuli (causes) with their associated outputs (effects). The graph is a result of an analysis of requirements. Test cases can be designed from the cause-effect graph.

The technique is a semiformal way of expressing certain requirements, namely requirements that are based on Boolean expressions.

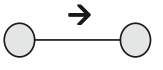
The cause-effect graphing technique is used to design test cases for functions that depend on a combination of more input items.

In principle any functional requirement can be expressed as:

***f*(old state, input) → (new state, output)**

This means that a specific treatment (*f* = a function) for a given input transforms an old state of the system to a new state and produces an output.

We can also express this in a more practical way as:

***f*(ops1, ops2,…, i1, i2,..i) → (ns1, ns2,..,o1, o2..)** 

where the old state is split into a number of old partial states, and the input is split into a number of input items. The same is done for the new state and the output.

The causes in the graphs are characteristics of input items or old partial states. The effects in the graphs are characteristics of output items or new partial states.

Both causes and effects have to be statements that are either True or False. True indicates that the characteristic is present; False indicates its absence.

The graph shows the connections and relationships between the causes and the effects.

Copyright © 2008. Artech House. All rights reserved.

Book: samlet.indb 169
Mette Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:42 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

4.1.4.1 Cause-Effect Graph Coverage

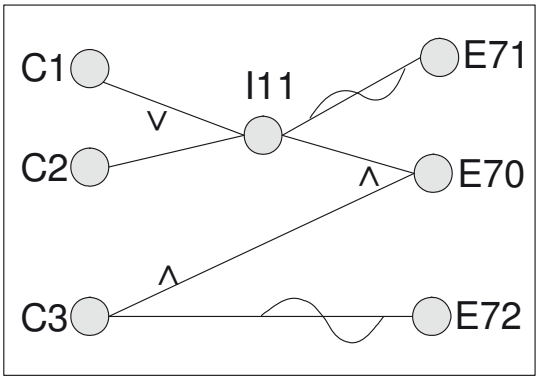
The coverage of the cause-effect graph can be measured as the percentage of all the possible combinations of inputs tested in a test suite.

4.1.4.2 Cause-Effect Graphing Process and Template

A cause-effect graph is constructed in the following way based on an analysis of selected suitable requirements:

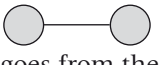
- List and assign an ID to all causes
- List and assign an ID to all effects
- For each effect make a Boolean expression so that the effect is expressed in terms of relevant causes
- Draw the cause-effect graph

An example of a cause-effect graph is shown here.



The graph is composed of some simple building blocks:

 Identified cause or effect—Must be labeled with the corresponding ID. It is a good idea to start the IDs of the causes with a C and those of the effects with an E. Intermediate causes may also be defined to make the graph simpler.

 Connection between cause(s) and effect—The connection always goes from the left to the right.

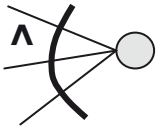
^ This means that the causes are combined with AND, that is all causes must be True for the effect to be True.

v This means that the causes are combined with OR, that is only one cause needs to be True for the effect to be True.

Copyright © 2008. Artech House. All rights reserved.

Book-sample.indb 170 2/19/08 8:14:42 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

This is a negation, meaning that a True should be understood as a False, and vice versa.



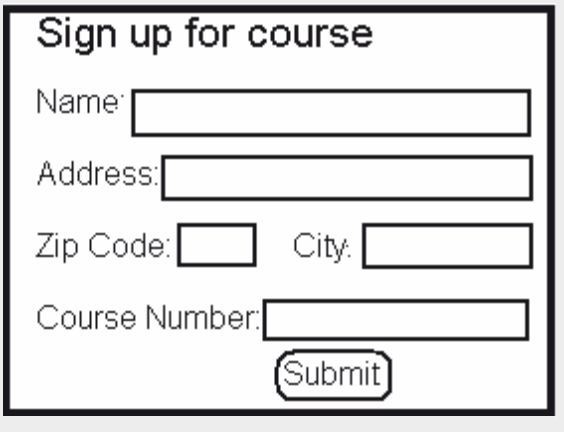
The arch shows that all the causes (to the left of the connections) must be combined with the Boolean operator; in this case the three causes must be "AND'et."

Test cases may be derived directly from the graph. The graph may also be converted into a decision table, and the test cases derived from the columns in the table.

Sometimes constraints may apply to the causes and these will have to be taken into consideration as well.

4.1.4.3 Cause-Effect Graph Example

In this example we are going to test a Web page, on which it is possible to sign up for a course. The Web page looks like this:



First we make a complete list of causes with identification. The causes are derived from a textual description of the form (not included here):

C1. Name field is filled in
C2. Name contains only letters and spaces
C3. Address field is filled in
C4. Zip code is filled in
C5. City is filled in
C6. Course number is filled in
C7. Course number exists in the system

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 171
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.
2/19/08 8:14:42 PM

An intermediate Boolean may be introduced here, namely I30 meaning that all fields are filled in. This is expressed as:

I30 = and (C1, C3, C4, C5, C6)

The full list of effects with identification is:

E51. Registration of delegate in system

E52. Message shown: All fields should be filled in

E53. Message shown: Only letters and spaces in name

E54. Message shown: Unknown course number

E55. Message shown: You have been registered

We must now express each effect as a Boolean expression based on the causes. They are:

E51 = and(I30, C2, C7)

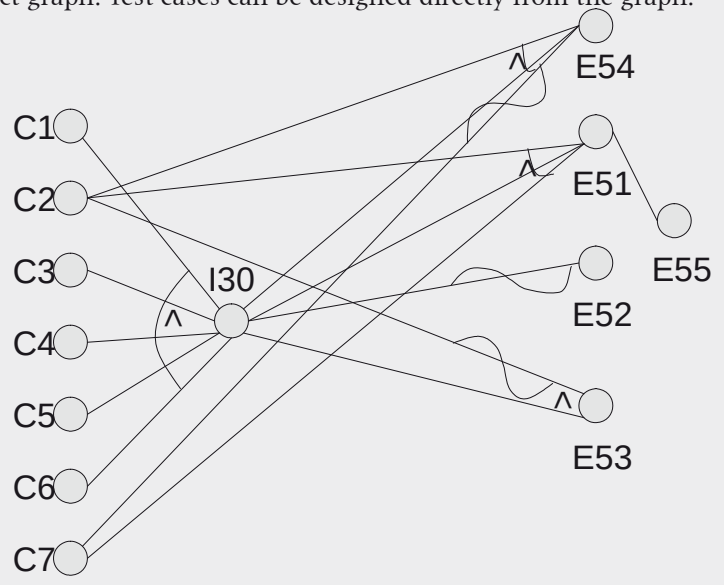
E52 = not I30

E53 = and(I30, not C2)

E54 = and(I30, C2, not C7)

E55 = E51

Drawing the causes and the effects and their relationships gives us the cause-effect graph. Test cases can be designed directly from the graph.



4.1.4.4 Cause-Effect Graph Hints

The cause-effect graph test case design technique is very suitable for people with a graphical mind.

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 172 2/19/08 8:14:43 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

For others it may be a help in the analysis phase and the basis for the construction of a decision table from which test cases can be described as discussed above.

All the effects can be filled into the decision table by looking at the cause-effect graph or even from the Boolean expressions directly. Fill in all combinations of True and False for the causes, and then fill in the impact each combination has on the effects.

Cause-effect graphs frequently become very large—and therefore difficult to work with. To avoid this divide the specification into workable pieces of isolated functionality.

To mitigate the size problem we can select different ways to reduce the decision table, such as removal of impossible combinations. We can also reduce the combinations by only keeping those that independently affect the outcome, like in condition determination testing, discussed in Section 4.2.6.

4.1.5 State Transition Testing

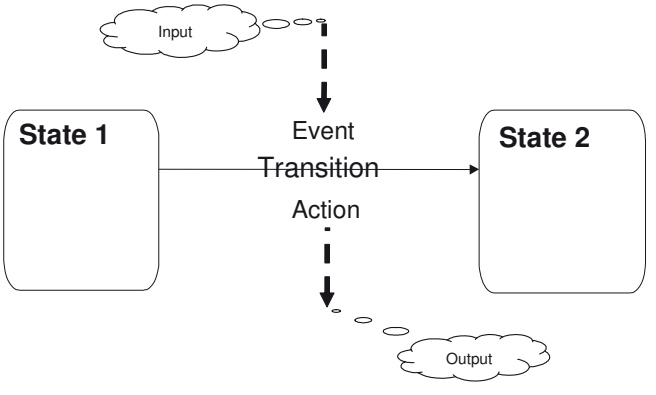
State transition testing is based on a state machine model of the test object. State machine modeling is a design technique, most often used for embedded software, but also applicable for user interface design. If we are lucky we get the state machine model as part of the specification we are going to test against. Otherwise we will have to extract it from the requirements.

Most products and software systems can be modeled as a state machine. The idea is that the system can be in a number of well-defined states. A state is the collection of all features of the system at a given point in time, including all visible data, all stored data, and any current form and field.

The transition from one state to another is initiated by an event. The system just sits there doing nothing until an event happens. An event will cause an action and the object will change into another state (or stay in the same state).

A transition = start state + event + action + end state

The principle in a state machine is illustrated next.



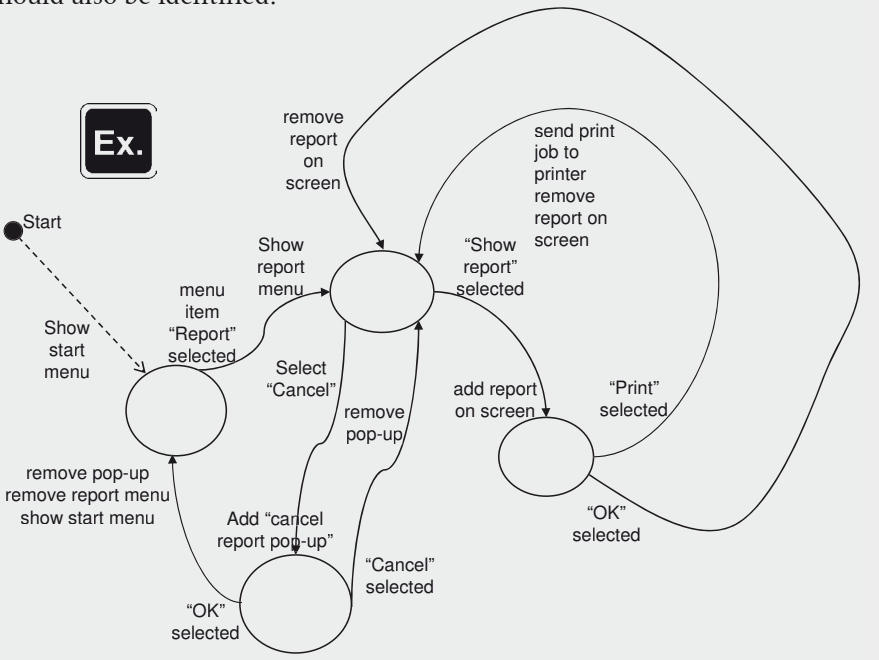
Copyright © 2008. Artech House. All rights reserved.

Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

Book: samlet.indb 173 2/19/08 8:14:43 PM

State machines can be depicted in many ways. The figure below shows a state machine presentation of a report printing menu, where the states are depicted as circles and the events and actions are written next to the transitions depicted as arrows.

It is a good idea to leave the states blank and just give them an identification, typically in the form of a number. The events, actions, and transitions should also be identified.



Note that the state machine has a start state. This could be a transition from another state machine describing another part of the full system.

4.1.5.1 State Transition Testing Coverage

Transitions can be performed in sequences. The smallest "sequence" is one transition at a time. The second smallest sequence is a sequence of two transitions in a row. Sequences can be of any length.

The coverage for state transition testing is measurable for different lengths of transition sequences. The state transition coverage measure is:

Chows n-switch coverage

where n = sequential transitions – 1.

We could also say that N = no. of "in-between-states."

Chows n-switch coverage is the percentages of all transition sequences of

Copyright © 2008. Artech House. All rights reserved.

Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

n-1 transitions' length tested in a test suite.

State transition testing coverage is measured for valid transitions only. Valid transitions are transitions described in the model. There may, however, also be invalid, or so-called null-transitions and these should be tested as well.

4.1.5.2 State Transition Testing Templates

A number of tables are used to capture the test conditions during the analysis of state transition machines.

To obtain Chows 0-switch coverage, we need a table showing all single transitions. These transitions are test conditions and can be used directly as the basis for test cases. A simple transition table is shown next.

The fields in the table are:

Test design item number: Unique identifier of the test design item
Traces: References to the requirement(s) or other descriptions covered by this test design
Assumptions: Here any assumption must be documented
The table must have a column for each of the defined transitions (three are shown here). The information for each transition must be:
Transition: The identification of the transition
Start state: The identification of the start state (for this transition)
Input: The identification or description of the event that triggers the transition
Expected output: The identification or description of the action connected to the transition
End state: The identification of the end state (for this transition)

Test design item number:			**Traces:**
Assumptions:			
Transition			
Start state*			
Input			
Expected output			
End state*			

* the "start"and "end" states are for each specific transition (test condition) only, not the state machine

Testing to 100% Chows 0-switch coverage detects simple faults in transitions and outputs.

To achieve a higher Chows n-switch coverage we need to describe the sequences of transitions.

The table to capture test conditions for Chows 1-switch coverage is shown next.

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 175
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.
2/19/08 8:14:44 PM

Test design item number:			Traces:
Assumptions:			
Transition pair			
Start state*			
Input			
Expected output			
Intermediate state*			
Input			
Expected output			
End state*			

Here we have to include the intermediate state and the input to cause the second transition in each sequence. Again we need a column for each set of two transitions in sequence.

If we want an even higher Chows n-switch coverage we must describe test conditions for longer sequences of transitions.

As mentioned above we should also *test invalid transitions.* To identify these we need to complete a state table. A state table is a matrix showing the relationships between all states and events, and resulting states and actions.

A template for a state table matrix is shown below. The matrix must have a row for each defined state and a column for each input (event). In the cross-cell the corresponding end state and actions must be given.

An invalid transaction is defined as a start state where the end state and action is not defined for a specific event. This should result in the system staying in the start state and no action or a null-action being performed, but since it is not specified we cannot know for sure.

Input / Start state	
	End state / Action

The "End state / Action" for invalid transitions must be given as the identification of the start state / "N" or the like.

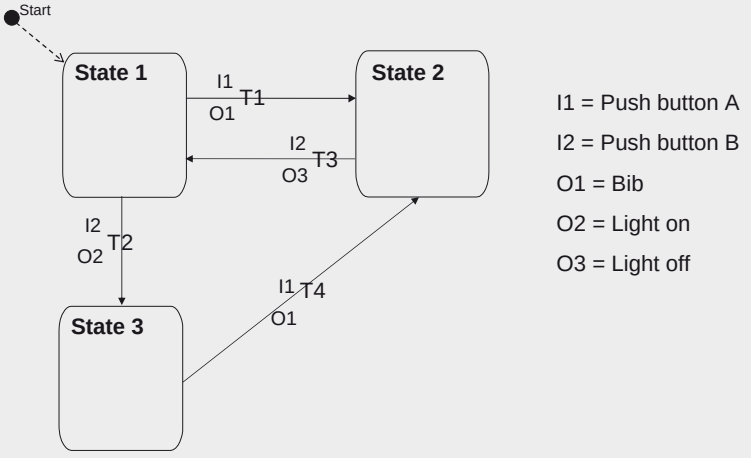
A test condition can be identified from this table for each of the invalid transitions.

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 176 2/19/08 8:14:45 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.

4.1.5.3 State Transition Testing Example

In this example we are going to identify test conditions and test cases for the state machine shown here.



Don't worry about what the system is doing—that is not interesting from a testing point of view.

The drawing of the state machine shows the identification of the states, the events (inputs), the actions (outputs), and the transitions. The descriptions of the inputs and outputs are given to the right of the drawing.

First of all we have to define test conditions for all single transitions to get Chows 0-switch coverage.

Test design item number: 2.4	**Traces:** State machine 1.1			
Assumptions: None				
Transition	T1	T2	T3	T4
Start state*	S1	S1	S2	S3
Input	I1	I2	I2	I1
Expected output	O1	O2	O3	O1
End state*	S2	S3	S1	S2

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 177
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.
2/19/08 8:14:45 PM

Identification of sequences of two transitions to achieve Chows 1-switch coverage results in the following table.

Test design item number: 2.5		**Traces:** State machine 1.1			
Assumptions: None					
Transition pair	T1/T3	T1/T3	T3/T2	T2/T4	T4/T3
Start state*	S1	S2	S2	S1	S3
Input	I1	I2	I2	I2	I1
Expected output	O1	O3	O3	O2	O1
Intermediate state*	S2	S1	S1	S3	S2
Input	I2	I1	I2	I1	I2
Expected output	O3	O1	O2	O1	O3
End state*	S1	S2	S3	S2	S1

We will not go further in sequences.

The next thing will be to identify invalid transitions. To do this we fill in the state table. The result is:

Input / **Start state**	I1	I2
S1	S2/O2	S3/O2
S2	S2/**N**	S1/O3
S3	S2/O1	S3/**N**

We have two invalid transitions:

State 2 + Input 1 and State 3 + Input 2.

The test cases to execute can now be created from these test conditions and written directly into a test procedure.

Test procedure: 3.5			
Purpose: This test procedure tests single valid and invalid transitions. Traces: State machine 1.1			
Prerequisites: The system is in State 1			
Expected duration: 5 minutes			
Tag	TC	Input	Expected output
T1	2	Reset to state 1 + push button B	The system bibs + state 2
T2		Reset to state 1 + push button B	The light is on + state 3
IV2		Push button B again	Nothing changes
T4		Push button A	The system bibs + state 2
IV1		Push button A again	Nothing changes
T3		Push button B	The light is off + state 1

Copyright © 2008. Artech House. All rights reserved.

Book: samiet.indb 178
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.
2/19/08 8:14:46 PM

4.1.5.4 State Transition Testing Hints

We should try to avoid invalid transitions by defining the results of invalid events. This is called defensive design, and it is a design activity.

If it is not practical to define all possible state and event combinations explicitly, we should encourage the designers to define a default for truly invalid situations. It could for example be defined that all null-transitions should result in a warning.

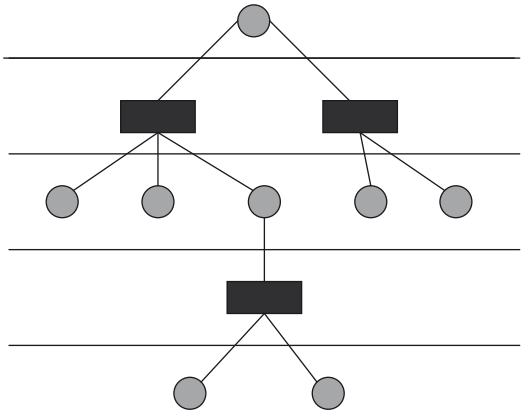
One of the difficulties of state machine is that they can become extremely complex faster and more often than you imagine. State machines can be defined in several levels to keep the complexity down, but this is a design decision.

4.1.6 Classification Tree Method

The classification tree method is a way to partition input and state domains into classes. The method is similar to equivalence partitioning, but can handle more complex situations where input or output domains can be looked at from more than one point of view.

The idea in the classification tree method is that we can partition a domain in several ways and that we can refine the partitions in a stepwise fashion. Each refinement is guided by a specific aspect or viewpoint on the domain at hand.

The result is a classification tree like the one shown here.



There are two types of nodes in the tree

- (Sub)domain nodes
- Aspect nodes

The two types must always alternate.

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 179
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:46 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

The domain is the full collection of all possible inputs and states at any given level in the tree. The state is a very broad term here; it means anything that characterizes the product at a given point in time and includes for example which window is current, which field is current, and all data relevant for the behavior both present on the screen and stored "behind the screen."

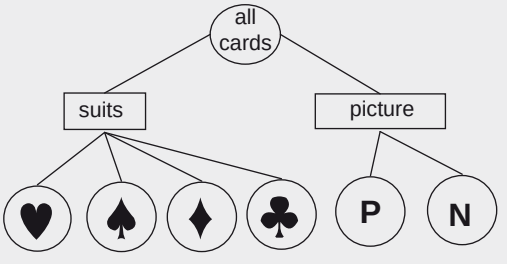
The aspect is the point of view you use when you are performing a particular partitioning of the domain you are looking at. It is very important to be aware that it is possible to look at the same domain in different ways (with different aspects) and get different subdomains as the result. This is why there can be more aspects at the same level in the classification tree and more subdomains at the same level (under the aspects) as well.

Ex.

An ordinary deck of cards (without jokers) can be viewed and hence partitioned in different ways. The aspects could be:

- Suit (spades, hearts, diamonds, clubs)
- Picture or number

The classification tree resulting from this analysis would look like this:



There are a few rules that need to be observed when we make the classification tree. Under a given aspect:

- *Any* member of the domain must fit into one and only one subdomain under an aspect—it must not be possible to place a member in two or more subdomains
- *All* the members of the domain must fit into a subdomain—no member must fall outside the subdomains

Ex.

If we look at the "suit" aspect above, no card can be in two suits, and all cards belong to a suit.

When we create a classification tree we start at the root domain (which is highest in the graph!). This is always the full input and state domain for the item we want to examine. We must then:

Copyright © 2008. Artech House. All rights reserved.

Book: samfet.indb 180
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:47 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

- Look at the domain and decide on the views or aspects we want to use on the domain
- For each of these aspects
  - Partition the full root domain into classes. Each class is a subdomain.
  - For each subdomain (now a full domain in its own right)
    - Decide aspects that will result in a new partitioning
    - For each aspect
    - And so on

At a certain point it is no longer possible or sensible to apply aspects to a domain. This means that we have reached a leaf of the tree. The tree is finished when all our subdomains are leaves (the lowest in the graph!).

Leaves can be reached at different levels in the classification tree. The tree does not have to be symmetric or in any other way have a predictable shape.

A leaf in a classification tree is similar to a class in an equivalence partitioning: We only need to test one member, because they are all assumed to behave in the same way.

4.1.6.1 Classification Tree Method Coverage

The coverage for a classification tree is the percentage of the total leaf classes tested in a test suite.

Leaves belonging to different aspects can be combined, so that we can reach a given coverage with fewer test cases. In areas of high risk we can also choose to test combinations of leaf classes.

4.1.6.2 Classification Tree Method Test Design Template

It is usually more practical to present a classification tree in a table rather than as a tree. A template for such a table where the test conditions are captured, is shown below.

Test design item number.:				**Traces:**				
Assumptions:								
Domain 1	**Aspect 1**		**Domain n**	**Aspect n**	**Tag**	**Tc1**		**Ten**

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 181
Mette, Jonassen Hass, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:47 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

The fields in the table are:
Test design item number: Unique identifier of the test design item
Traces: References to the requirement(s) or other descriptions covered by this test design
Assumptions: Here any assumption must be documented.
Domain 1: A description of the (root) domain
Aspect 1: A list of the aspects defined for domain 1
For each aspect a list of subdomains are made.
For each of the subdomains new aspects are identified or the subdomain is left as a leaf.
This goes on until we have reached the leaves in all branches.
Tag: Unique identification of the leaves = test conditions
Tc1: A marking of which test cases cover the test conditions

4.1.6.3 Classification Tree Method Example

In this example we are going to test the following requirements for a small telephone book system:

(1) A person can have more than one phone number
(2) More than one person can have the same phone number
(3) There is one input field where you can type either:

- Phone number
- All names in the full name
- Some of the names in the full name

(4) A person shall be found if one or more names match
(5) One or more people shall be found if the phone number matches
(6) The output shall be either:

- An entry for each person that is found
- An error message: no person found

First we fill in the header:

Test design item number: 56	**Traces:** Req. (1) – (6)
Assumptions: None	

The next thing is to look at the root and identify the first level of aspects:

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 182
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
2/19/08 8:14:47 PM
Created from ncent-ebooks on 2021-05-29 19:59:36.

Domain 1	Aspect 1
All inputs and types of lists	
	input type
	match?
	state of list
	result

Now we must take each of these aspects one by one and filter the root domain through them. The leaf subdomains are marked in bold.

Domain 1	**Aspect 1**	**Domain 2**
All inputs and types of lists		
	input type	**pure text**
		pure number
		mixture
		empty
	match?	**no**
		yes
	state of list	**empty**
		not empty
	result	**nothing found**
		something found

We now have seven subdomains that are leaves, and three that can be further broken down by new aspect. So we find some aspects for each of the remaining subdomain, and find the subdomain for each.

The "old" leaves have been left out of the table for the time being, and the "new" leaves are marked in bold.

Domain 1	**Aspect 1**	**Domain 2**	**Aspect 2**	**Domain 3**
All inputs and types of lists				
	match?	yes	*no.of matches*	**1**
				more than 1
			type of match	name
				phone no.
	state of list	not empty	*1 name + 1 no.*	**yes**
				no

Copyright © 2008. Artech House. All rights reserved.

Book: sample.indb 183
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.
2/19/08 8:14:48 PM

Domain 1	Aspect 1	Domain 2	Aspect 2	Domain 3
All inputs and types of lists				
	match?	yes	*no.of matches*	**1**
				more than 1
			type of match	name
				phone no.
	state of list	not empty	*1 name + 1 no.*	**yes**
				no
			1 name + many no	**yes**
				no
			many names + 1 no.	**yes**
				no
			1 name + 1 no.	**yes**
				no
	result	something found	*1 name + many no.*	**yes**
				no
			many names + 1 no.	**yes**
				no

Now we have only one subdomain, which is not a leaf left to deal with, namely a match of the name. The aspect to use here is how much of the name matches, and the subdomains are: "full name" and "part of name." These are leaves.

Before we go any further we have to control the partitioning. We need to ensure that any member of a domain fits into one and only one subdomain.

In this example we find that a string of blanks belongs to more than one subdomain, because a blank is considered to be text. To resolve that we require that pure text contains at least one letter.

Our test conditions are now defined, and we must go on defining our test data and test cases.

If we examine the classification tree we can see that we need a number of phone lists as our test data.

Copyright © 2008. Artech House. All rights reserved.

Book-sample.indb 184 2/19/08 8:14:48 PM
Mette, Jonassen Haas, Anne. Guide to Advanced Software Testing, Artech House, 2008. ebrary, Quest Ebook Central, http://ebookcentral.proquest.com/lib/ncent-ebooks/detail.action?docID=456922.
Created from ncent-ebooks on 2021-05-29 19:59:36.