The Testing Process – A Decision-Based Approach

Technical Report SWEHD-TR-2007-01

Lars Borner, Universität Heidelberg, Arbeitsgruppe Software Systeme
Timea Illes-Seifert, Universität Heidelberg, Arbeitsgruppe Software Systeme
Barbara Paech, Universität Heidelberg, Arbeitsgruppe Software Systeme

Version 1.0
September, 2007

Eine Publikation der Arbeitsgruppe Software Engineering

Historie

Version	Datum	Grund der Änderung
V.1.0	September 2007	Ersterstellung

Die Arbeitsgruppe „Software Engineering" ist Teil
des Instituts für Informatik der
Ruprecht-Karls-Universität Heidelberg.
Sie wird geleitet von
Prof. Dr. Barbara Paech

Institut für Informatik
Neuenheimer Feld 348
69120 Heidelberg
paech@informatik.uni-heidelberg.de
http://www-swe.informatik.uni-heidelberg.de/

The Testing Process - A Decision-Based Approach

Lars Borner, Timea Illes-Seifert, Barbara Paech
Universität Heidelberg, Im Neuenheimer Feld 326, 69120
{borner, illes, paech}@informatik.uni-heidelberg.de

Abstract

Considering that testing a software system completely is not possible, the main task of a test team is to decide which parts of a system should be tested in which way. The numerous decisions are usually made implicitly during the testing process. However, awareness of these decisions increases their quality, by forcing the decision-makers to search for alternatives and to trade off between them. In this technical report we propose a decision hierarchy for the testing process. This hierarchy comprises the decisions made during testing and reflects dependencies among them. These decisions can be assigned to several decision levels as well as to different roles involved in the testing process, resulting in a decision hierarchy. The decision hierarchy and the identified decisions can be applyied in different contexts. In this report, we additionally present the results of four case studies to which we applied this decision hierarchy.

1 Introduction

Today's software systems consist of numerous software components; they realize countless requirements and are developed in an industrial environment limited by high time and resource constraints. In order to assess to which extent a software system or its parts fulfill the requirements, testing activities have to be performed. Since complete testing is impossible [16], testers are forced to make decisions, i.e. to decide which parts of the software system have to be tested in which way. Usually, these decisions are made implicitly by the corresponding roles and often, the responsible persons are not aware of the decisions they made. However, the awareness of decisions can significantly improve their quality as it restricts the infinite possibilities to test a system to a finite set of test cases. Making a decision consciously forces the person who has to take this decision to search for alternatives, to establish selection criteria and to trade off between advantages and disadvantages of several alternatives. Consequently, the awareness of decisions leads to better decisions compared to implicit or ad hoc decisions and increases the quality of the testing process.

In this paper, we define a decision as follows: *A decision denotes a choice consciously or unconsciously made by a person or a group of persons. A decision made consciously evolves in the process of discussing possible alternatives and considering existing success criteria.* During the software development process as well as during the testing process, several decisions have to be made. The best alternative has to be selected from e.g. alternative GUI designs, architectural patterns or testing techniques.

In this technical report we identify these decisions and the corresponding roles of the testing process. Moreover, we assign each decision to one of seven decision levels which lead to a decision hierarchy. This decision hierarchy can be used for different purposes, e.g. as a framework to categorize existing testing approaches or as a framework to analyze actual testing processes in an organization and to derive process improvements. Furthermore, the decisions of the decision hierarchy can be applied as a checklist to plan testing activities.

In our research work we applied the decision hierarchy in four case studies. We specialized the hierarchy to identify specific decisions to be made during the system and integration testing process to highlight the differences and commonalities of both processes. Furthermore, we applied the hierarchy to analyze different testing processes in the industry in order to uncover strengths and weaknesses of the executed processes and to develop possible improvements. Within another research context the hierarchy was applied to classify different testing techniques in order to identify the supported decisions of the considered techniques. Finally the hierarchy served as the basis of a questionnaire for a test tool evaluation.

The remainder of this report is organized as follows. Section 2 gives an overview of related work and Section 3 describes the generic decision hierarchy for the testing process, containing decision levels, corresponding decisions and roles. Section 4 presents results of four case studies, to which we applied this decision hierarchy and Section 5 gives a short summary and discusses the results of our approach.

2 Related Work

A process model, which describes the main phases of the testing process, consisting of test planning, test design, test execution and test evaluation activities has been proposed in [22] by Spillner, Linz and Schäfer. In comparison to our approach, which explicitly focuses on all decisions to be made during the testing process, the process model described in [22] is very generic and does not take into account the decisions involved. The IEEE standard for software test documentation [10] specifies all artifacts to be created during the testing process, e.g. test plan, test design specification, test case specification. The decisions made in the testing process are not part of the standard. Another group of related work comprises test process improvement models like TPI (Test Process Improvement) [15] or test maturity assessment models, e.g. TMM (Testing Maturity Model) [5]. The focus of these models is not the test process itself, but the steps for its improvement, respectively the criteria to assess the maturity of the organizational testing process.

A conceptual framework categorizing different decisions made during requirements engineering has been presented in [19] by Paech et al. and in [2] by Aurum et al., but these approaches do not consider decisions to be made during other phases of the software engineering process. Furthermore, the system Sysiphus, supporting the documentation of decisions defined in [19], has been realized in [24]. Additionally, several approaches for the documentation of the decisions made during the software development process have been proposed in [9]. To the best of our knowledge there is no existing research and there are no case studies which particularly address the decision making process in quality assurance activities.

3 Decision Hierarchy

In our research work we identified the decisions to be made during the testing process and assigned them to decision levels. At first, we identified the tasks and roles by analyzing test process descriptions mentioned in standard textbooks such as Spillner [22] and Mosley and Posey [17]. Our work is mainly based on the fundamental test process described in [22] consisting of test planning and specification, test execution, capturing and analysing test results. In a next step, we identified decisions to be made while performing testing tasks and grouped them into seven decision levels. The result is the generic decision hierarchy illustrated in Figure 1.

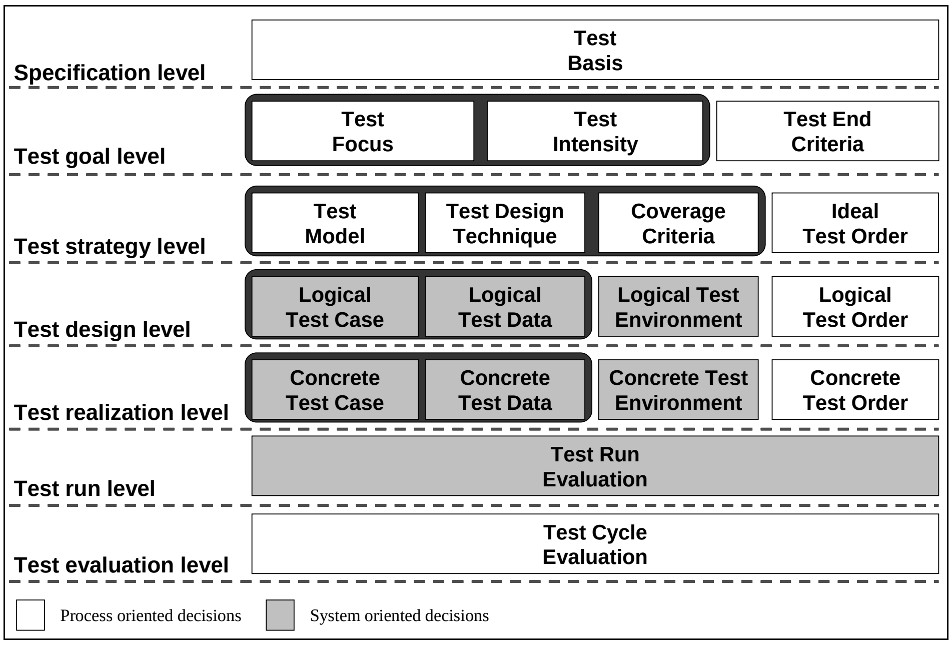

Figure 1. Decision levels and corresponding decisions of the testing process

The *principles* behind the decision hierarchy can be defined by the following rules:

R1 Decision dependencies: Decisions at lower levels depend on decisions made at earlier levels. If decisions at top levels are left out, they are implicitly contained in decisions made at lower levels. Leaving out a decision decreases the quality of this particular decision, as well as the quality of all dependent ones. The goal of making decisions in the proposed order is to facilitate the decision making process.

R2 Parallelism: All decisions on the same level can be done in parallel, i.e. these decisions can be made nearly independently, but they may influence each other. Decisions that influence each other can be combined to *decision bundles*. In Figure 1 decision bundles are represented by a dark grey box behind the corresponding decision (e.g. test focus and test intensity belong to one and the same bundle).

Moreover, two different *perspectives* on the decisions can be identified. One perspective contains decisions which influence the testing process (called process-oriented decisions), i.e. which test artifacts will be created. Another perspective contains decisions concerning the system under test (called system-oriented decisions), i.e. how the system will be tested. The top level decisions try to give answers to the question *which parts* of the software system have to be tested. For this purpose the roles responsible for decisions on this level use information contained in the specification of the system (e.g. in the requirements specification, in the architecture or design specification). The subsequent levels make decisions on *how* the (parts of the) system should be tested. The lower the level, the better the tester can specify how the parts of the system should be tested. On the last two levels, decisions concerning the *evaluation* of the test runs have to be made.

In the following, we first introduce important roles of the testing process followed by a detailed description of the decision hierarchy.

3.1 Testing Roles

Since the testing process consists of a series of activities, roles responsible for these different activities have to be assigned. In our research work we distinguish between to categories of roles: testing roles and test supporting roles. Testing roles subsume all roles directly involved in the testing process which are responsible for decisions to be made during the testing process. The test manager, the test designer and the tester are typical testing roles [22]. Test supporting roles subsume all roles of the software development process which deliver information essential to make decisions during the testing process. E.g. the requirements engineer and the system architect are typical test supporting roles.

The **test manager** is responsible for activities such as planning and controlling of the testing process. This includes resource planning and scheduling as well as risk analysis activities. It is his/her task to enable and to ease the activities of other testing roles during the process. Furthermore, he/she has to decide which parts of the software have to be tested in which way. At the end of the testing process he/she decides whether the test activities were successful and whether they can be finished or not. The test manager communicates especially with the test designer and with the testers as well as with the project manager of the software development project. The test manager makes decisions mainly on the test goal and test strategy level as well as on the test evaluation level.

The **test designer**, on the one hand, supports the decisions of the test manager, i.e. he/she helps to select the parts of the software to be tested or the required testing strategy to be applied. On the other hand he/she is responsible for the design of the test cases. Therefore, he/she applies different test design techniques to derive test cases on the basis of the specification of the software system to be tested. The test designer is involved in nearly all activities of the testing process, except the test planning and controlling tasks. Therefore, he/she intensively communicates with the test manager and the testers, but also with the supporting testing roles (e.g. with requirement engineers, system architects, system designers, programmers, and with the project manager) to gather information on the system under test.

The main task of the **tester** is to realize the test design defined by the test designer. His/Her main activities subsume the execution of test cases and the logging of the results of a test run. In the case of automation the tester is responsible for the development of the test automation scripts and of the test environments. The main communication partner of the tester is the test designer, but the tester also communicates with some test supporting roles, e.g. with requirement engineers, system designers or programmers.

3.2 Specification Level

The specification level is the top level of the decision hierarchy and contains decisions which deal with the completeness of the **test basis**. The test basis includes all information needed for a successful start of the testing process and often consists of the specification of the software system at different development stages (e.g. requirements specification or system design specification). The test basis defines a set of test objects, their behavior, their input and their output as well as the specification of possible dependencies between the test objects. We presume the definition in [11] of a software system, including its specification as well as its implementation (represented by code) and define a test object to be a part of a software system. At specification level it has to be decided, whether the test basis is complete or not. If information in the test basis is missing, the test designer has to complete the given specification. To gain an exhaustive test basis he/she cooperates with the creator of the test basis (e.g. requirement engineer, system architect, system designer, programmer, etc.) to identify and complete the missing information. Missing information in the test basis can lead to the fact that critical parts of the software are overlooked and thus remain untested. The decisions on this level influence nearly all decisions on the lower levels. They are the basis for the selection of the test foci and the test intensity, as well as the basis for decisions on the testing strategy, the test design and the realization.

3.3 Test Goal Level

Considering that a software project usually is limited in time, not all parts of the test basis can be tested. Therefore, at test goal level the test manager and the test designer have to decide which parts of the system have to be tested and which not. For this purpose, it is essential to possess a complete test basis in order to select the critical test objects. If some important information is missing in the test basis and can consequently not be considered in the decisions, critical parts of the software can be overlooked and thus remain untested. This may lead to a dangerous situation in which the final software release contains critical defects.

We denote all parts of the system which have been selected to be tested as **test foci**. Usually, the test foci represent all critical parts of the test basis. Critical in this context means, e.g. that the corresponding parts of the software will be used frequently during run time, that they will cause high damage (to the user, to the software system or to the environment) if they fail, that they are very complex so that the probability to fail is high or that they may contain already known defects.

Besides time pressure within the testing process another constraint influences the decisions on this decision level: cost. The cost constraints lead to a limitation of resources needed within the testing process, e.g. the number of test designers and testers or the budget for hardware, software or for staff training. Therefore, the existing resources have to be split up among several test foci. To grant the correct assignment of resources to the various test foci, it has to be decided which **test intensity** (measured e.g. by man days or funds) has to be assigned to a single test focus, i.e. how intensively a single test focus will be tested. Therefore, the test intensity serves as an indicator for the required test effort per focus and is used by the test manager to allocate resources to the test foci.

The decisions on test intensity and test focus influence each other and consequently belong to a bundle. Decisions on the **test end criteria** can be made independently from this bundle. The test end criteria define conditions which have to be fulfilled to finish the testing activities, e.g. they can give information about the required rate of successful test runs. The test manager is responsible for the selection and definition of these test end criteria.

3.4 Test Strategy Level

After the test foci and the test intensities have been identified, the test manager and test designer decide on the test strategy to be used in the testing process. The test strategy comprises decisions related to the test design techniques, the test model(s) and its coverage(s) as well as the ideal test order. One decision to be made concerns the **test design technique** which will be used to derive test cases and test data from the test basis. For each test level (system, integration and unit test level) a countless number of test design techniques can be found in the literature (e.g. in [3], [4], [16], [22]). Therefore, existing test design techniques, the defined test foci and test intensities have to be taken into account in order to select the most adequate test design technique(s).

In parallel, decisions concerning the **test model** have to be made by the test designer. A test model facilitates the derivation of test cases and test data in comparison to the derivation from an informal specification. A state based model or a control flow model are examples of test models. The test design technique influences the selection of the test model and vice versa. Later in the testing process, the selected test design techniques have to be applied in order to derive test cases and test data to achieve the given test coverage and to fulfill the test **coverage criteria**. The test coverage is an indicator for the number of test cases to be derived. The test design technique influences the decision on coverage criteria and vice versa, e.g. if state based test design techniques are used to derive test cases, it will not be appropriate to define equivalence class coverage as test coverage indicators.

Furthermore, on this decision level an **ideal test order** to test the different test objects has to be specified. The ideal test order represents an optimal order to test the different parts of the system by taking into account the information on the test foci, on test intensity and on the coverage criteria. An example of such an ideal test order would be that all test objects with the highest test intensity should be tested first, followed by the ones with the next lowest intensity and so on.

3.5 Test Design Level

The test design level is the most important and most complex level of the testing process. On this level, the test designer decides how to apply the predefined test design techniques to reach the required test coverage. The main decision on this level is how to test the different test foci, i.e. the selected test objects. Therefore, the given test design techniques are applied to derive **logical test cases** (also called abstract test cases) [12], [22]. A logical test case gives an abstract description of how to test a specific aspect of the objects under test. A logical test case usually contains the description of a general goal (What is to be tested by this test case?), the required pre- and post-conditions (Which conditions have to be fulfilled so that this test case can be executed / after the execution of a test case?) and the various steps of this test case (Which steps have to be performed?). Additionally, the test case describes how test data have to be made available (e.g. input by a tester or input from a dedicated database) and how to observe the expected response of the test object [13]. Logical test cases do not describe a concrete action to be executed on the test object (e.g. "press the button with the label 'submit'"). They rather specify a more general action that should be executed (e.g. "user submits the input").

In parallel to the test case design, it has to be decided which **logical test data** serve as an input for the test objects within the test case. The logical test data represent the abstract description of the data to be sent to and returned by the test object. For example this could be the description of an equivalence class or a set of possible values ([3], [16], [22]). Both, the specification of a logical test case and the required test data, are connected. A logical test case without the required logical test data is not complete and vice versa.

The third decision on this level concerns the definition of the **logical test environment**. The test designer has to decide what kind of tools, software or hardware, is needed during the execution of the test cases. The description of the logical test environment is also abstract similar to the specification of the logical test cases or test data and represents the general requirements on the test environment. It illustrates the general requirements on the test system. E.g. the test designer decides that the execution of a test case needs a monitor to record the outcome of the test objects or to observe the inner communication of a set of test objects, but he/she does not specify which specific monitor is required.

The last decision at the test design level discussed here is related to the **logical test order**. This order refines the ideal test order considering dependencies between test cases as well as information about planned test environment factors. Execution efficiency and parallelism are the main criteria influencing this decision. A typical planned project environment factor which can lead to a changed ideal order is the planned completion time for the corresponding test object (realization), i.e. a test focus with high assigned test intensity cannot be tested as defined in the ideal test order before the corresponding test object has been implemented.

3.6 Test Realization Level

The test realization level details the logical representation of the test cases, of the test data as well as of the test environment. It contains all decisions which influence the execution of a test case. It contains all required decisions which affect the execution of a test case. Most of the decisions on this level are made by the tester with support of the test designer. This level comprises decisions on the concrete test order, on concrete test cases, concrete test data and the concrete test environment. Setting up the **concrete test order** means to identify an actual executable test order (*Which is the best possible order for the concrete test case execution?*) considering the logical test order and the project environment factors (*Which is the best order considering the actual project circumstances?*).

In parallel, the test refines logical test cases by **concrete test cases.** He/she adds information on the specific behavior of the test case and the test object. Concrete test cases contain all information needed to execute the test case. The concrete actions within every single test step are specified in detail, i.e. the tester specifies how to send test data to the test object (e.g. "insert the age into the input field labeled 'Age' and press the button labeled 'submit'"). Furthermore, the check actions of the test cases are described in more detail, i.e. the tester specifies all information needed to decide, if a test passes or fails including the specification of the expected response and how it can be verified.

To complete the specification of a concrete test case, the detailed description of the **concrete test data** is needed. Consequently, it has to be decided which concrete "instances" of the logical test data are used in the concrete test cases. The logical test data only give an abstract description of the data required by the test case. The task of the tester is now to choose concrete "instances" for the logical test data. Concrete instances of the logical test data can be representative values for an equivalence class or any other value that fulfils the conditions contained in the abstract description of the corresponding logical test data.

The last decisions on the test realization level are decisions on the **concrete test environment**. Here the tester has to take into account the description of the logical test environments and the specification of the logical test cases, in order to specify the concrete test environments for the test cases. The concrete test cases need a corresponding concrete test environment (e.g. the specification of concrete hardware and software needed) to be executable

After all decisions on this level have been made, the tester is able to realize the test cases and the test environment (e.g. by implementing the required test code), to execute the test cases (i.e. manually or automatically) and to record the results of the test run (e.g. by using monitors or test tools).

3.7 Test Run Level

The test run level deals with the evaluation of test run results. After the execution of a test case the tester has to decide, whether the test run was successful, that means whether the tested test objects have not shown the expected behavior and have not delivered the expected outcome. If this is the case, i.e. the test run was successful, the tester has to decide whether the test case actually revealed a defect in the realization of the test objects or not. In the latter case the defect can be found in the test case specification (logical or concrete), or in the realization of the test environment. To evaluate the test run, the tester needs the logs of the test run, including executed steps and actual test data (if available, former test logs can be used) and the specification of the expected result. By comparing the expected and actual results, the tester can decide whether a defect in a test object exists – consequently he/she performs a **test run evaluation**. In the case of a defect in the test object the test manager assigns a state (e.g. open), a priority (e.g. patch) and a weight (e.g. system crash) to a defect [22].

3.8 Test Evaluation Level

This level contains the decision whether the test activities can be finished. The decisions in the **test cycle evaluation** check whether the test end criteria have been fulfilled and whether every test focus has been tested with the required test intensity. Furthermore, the defects not found within this test cycle are estimated by using a metric like the defect detection rate. The decision not to finish the test cycle, leads to a new iteration of some (or maybe all) of the testing tasks and decisions. These decisions are made by the test manager.

4 Applying the Decision Hierarchy

In this section we apply our decision hierarchy to the system and integration testing in order to identify the specific issues and decisions of both processes. The purpose of this task is to identify the specific decisions in both testing processes by instantiating the generic decision hierarchy. Figure 2 sums up the main results of this instantiation. It illustrates all decision levels of the testing process as well as the decisions in the generic testing process (left column), the specific decisions in the system testing process (middle column) and the specific decisions in the integration testing process (right column). Specific decisions in the middle and the right columns refine corresponding decisions in the generic testing process at the same decision level. This is illustrated in Figure 2 by using two labels within one “decision box”. The upper label of a box describes the decision in the generic testing process. The lower label specifies the corresponding specific decision in the system testing process, respectively in the integration testing process.

In the following subsections we describe the specific decisions in the system and integration testing in more detail.

4.1 Evaluation Framework for the System and Integration Testing Processes

We applied our decision hierarchy to the system testing process (STP) and integration testing process (ITP) in order to identify the specific issues and decisions in both processes by instantiating the generic decision hierarchy. Figure 2 summarizes the main results. It illustrates all decision levels as well as the corresponding decisions in the generic testing process (left column), the specific decisions in the system testing process (middle column) and in the integration testing process (right column). Specific decisions in the middle and the right columns refine corresponding decisions in the generic testing process at the same decision level. This is illustrated in Figure 2 by using two labels in one “decision box”. The upper label of a box describes the decision in the generic testing process. The lower label specifies the corresponding specific decision in the STP, respectively in the ITP.

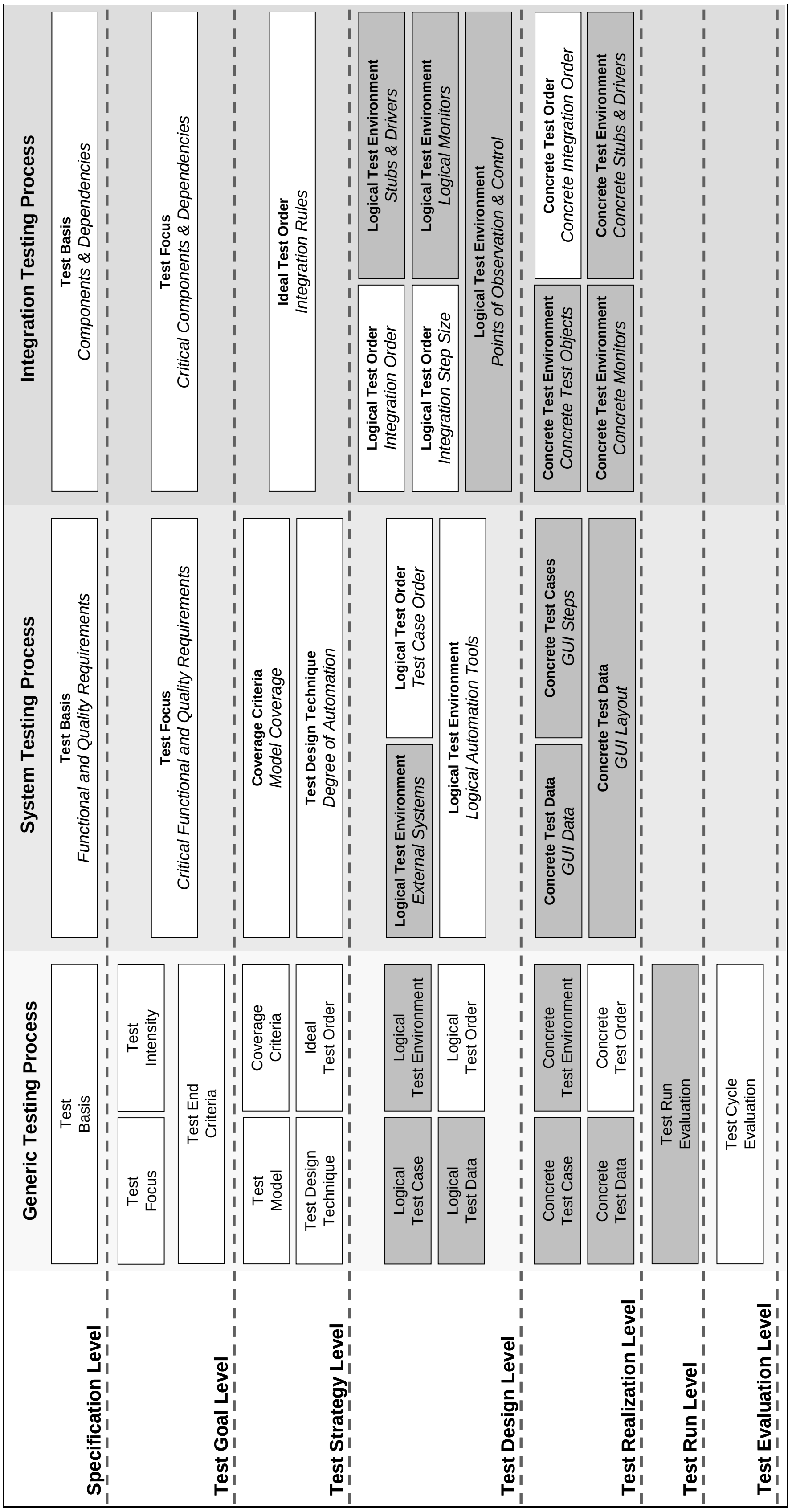
Specification Level
Test Goal Level
Test Strategy Level
Test Design Level
Test Realization Level
Test Run Level
Test Evaluation Level
Generic Testing Process
Test Basis
Test Focus
Test Intensity
Test End Criteria
Test Model
Coverage Criteria
Test Design Technique
Ideal Test Order
Logical Test Case
Logical Test Environment
Logical Test Data
Logical Test Order
Concrete Test Case
Concrete Test Environment
Concrete Test Data
Concrete Test Order
Test Run Evaluation
Test Cycle Evaluation
System Testing Process
Test Basis Functional and Quality Requirements
Test Focus Critical Functional and Quality Requirements
Coverage Criteria Model Coverage
Test Design Technique Degree of Automation
Logical Test Environment External Systems
Logical Test Order Test Case Order
Logical Test Environment Logical Automation Tools
Concrete Test Data GUI Data
Concrete Test Cases GUI Steps
Concrete Test Data GUI Layout
Integration Testing Process
Test Basis Components & Dependencies
Test Focus Critical Components & Dependencies
Ideal Test Order Integration Rules
Logical Test Order Integration Order
Logical Test Environment Stubs & Drivers
Logical Test Order Integration Step Size
Logical Test Environment Logical Monitors
Logical Test Environment Points of Observation & Control
Concrete Test Environment Concrete Test Objects
Concrete Test Order Concrete Integration Order
Concrete Test Environment Concrete Monitors
Concrete Test Environment Concrete Stubs & Drivers

Figure 2. Specific decisions in the system and integration testing processes

In the STP as well as in the ITP, decisions concerning the test basis and test focus are refined. Both testing processes deal with different kinds of information and specifications, e.g. functional and quality requirements in the STP and components and dependencies in the ITP in order to decide on the critical parts to be tested. On the test strategy level the integration testing process defines several integration rules to provide guidelines for the later integration test order. In the STP decisions on model coverage and the degree of automation refine the generic decisions.

At test design level both processes refine the decisions on the logical test environment and the logical test order. In the STP, the kind of external systems and the automation tools to be used in the test execution phase are chosen, whereas in the ITP decisions on the required stubs, drivers, monitors and the points of observation and control are made. In the ITP the focus of the test order lies on the integration order and the integration step size, i.e. the order and the number of components added in one integration step. On this level, the STP decisions on the optimal test case order minimizing the setup-overhead for the test cases play an important role.

At test realization level the STP refines the decisions on concrete test data and concrete test cases whereas the ITP deals with decisions on the concrete test environment and the concrete test order. Especially for the STP, decisions concerning GUI steps are important in order to define the concrete test cases. Moreover, GUI data is used to select concrete test data. In parallel, the GUI layout, i.e. how the GUI data is arranged on the screen, influences the concrete test cases. At test realization level, the ITP defines a concrete test order considering the real completion time of every component, the integration rules, order, and step size. Furthermore, decisions on how to prepare the test object (e.g. inserting points of control and observation), how to implement concrete monitors, stubs and drivers have to be made. On the last two decision levels there are no specific decisions in the STP and the ITP.

4.2 Evaluation Framework for Testing Tools

The decision hierarchy served as the basis for the definition of a questionnaire used in a survey evaluating 13 commercial and open source test management tools [14]. The evaluation is primarily based on the information provided by tool vendors in the questionnaire. The goal was to analyse to what extent a decision is supported by a test management tool. Based on the decision hierarchy, questions addressing the functional characteristics of the testing tool can easily be derived. E.g. if a test management tool integrates requirements management functionality, it would provide support for decisions on the specification level by facilitating the identification of functional and quality requirements.

In the following, we evaluate an open source testing tool in order to exemplify the use of the decision hierarchy as an evaluation framework for testing tools. Salomé TMF (Test Management Framework) [25] is an open source test management tool, which is developed by the ObjectWeb [26] consortium. Main goals of the consortium subsume the development of distributed, component based middleware. The project Salomé TMF has been registered in May 2005.

Salomé TMF supports the documentation of requirements by a plug-in. In a text box, a requirement can be specified and any attachments can be assigned. Thus, the decisions concerning the test basis are rudimentarily supported by the tool. None of the decisions on the test goal and test decision level are supported by Salomé TMF. A *test suite* subsumes a set of *tests* ordered in a sequence in a *test campaign.* A test contains several *actions* and *parameters*. Thus, the *documentation* of the decisions concerning the logical test cases (= tests), the logical test order (=test campaign) and the logical test data (= parameters) is supported by the test management tool. Parameters can be refined by concrete values (*data sets).* Additionally, Salomé supports the specification of concrete environment parameters (*environment parameters and values*) and of concrete test sequences (*execution*). Consequently, the tool supports the documentation of almost all decisions on the test design and test realization level. The specification of the expected and actual test results is possible, but Salomé does not support the definition of metrics which allow the evaluation of a test cylcle. The export to an external bug tracking system (Bugzilla [27]) is only possible, where a complex analysis of test results is possible.

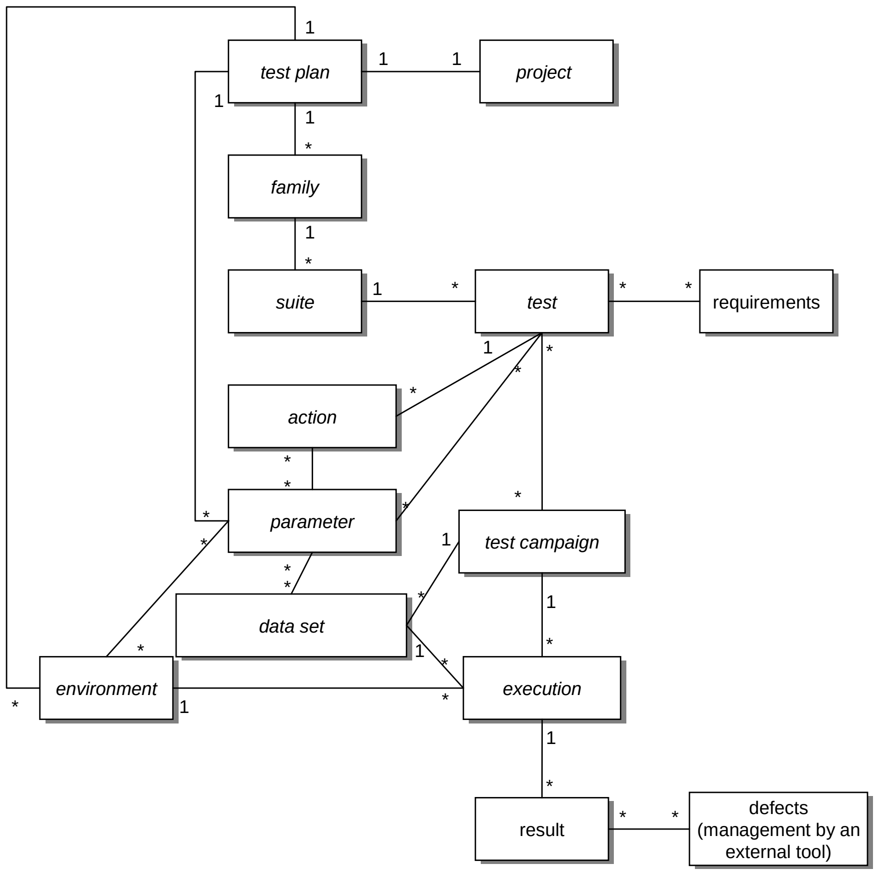

Figure 3. Salomé conceptual model

Table 1 represents the evaluation of the open source test management tool Salomé TMF. The test tool supports mainly the documentation of decisions on the test design and test realization level. Thus, this tool is useful mainly for test designers and testers. Decisions on the specification, test run and evaluation level are supported only rudimentarily. The tool does not support the decision making process itself, but the documentation and the management of the decisions and their resulting artefacts.

	Decisions	Tool
Decision level	**Generic tesing process**	**Salomé TMF**
Specification level	Test basis	(X)
Test goal level	Test focus	
	Test intensity	
	Test end criteria	
Test strategy level	Test model	
	Coverage criteria	
	Test design technique	
	Ideal test order	
Test Design Level	Logical Test Case	X
	Logical Test Environment	
	Logical Test Data	X
	Logical Test Order	X
Test Realization Level	Concrete Test Case	
	Concrete Test Environment	X
	Concrete Test Data	X
	Concrete Test Order	X
Test Run Level	Test Run Evaluation	(X)
Test Evaluation Level	Test Cycle Evaluation	(X)

Table 1 - Evaluation of Salomé TMF

4.3 Test Process Analysis

Based on our decision hierarchy, the testing process of an organisation was analyzed in order to find its strengths and weaknesses. The organisation we refer to provides system solutions in the area of remote operations. Testers in this organisation are organized in an independent testing group. The ratio of testers to developers is 1:4. The test process analysis is based on document reviews as well as on interviews. All interviewees are experienced testers, with up to ten years of experience. In the following we describe the results of our analysis.

All decisions at *specification level* are made by the requirements engineering team, whereas the rest of the decisions are made by the testing team. Furthermore, there are decisions made implicitly, e.g. all decisions at test goal and test strategy level and decisions made explicitly, e.g. all decisions at test design level. Implicit decisions are not documented, whereas explicit decisions are (partially) documented in test artefacts. All decisions on the test goal and test strategy level are made implicitly. The testing team does not perform a risk analysis in order to make sound decisions on test foci or test intensities. Thus, the end of the testing activities is not determined by criteria defined in advance, but by current test results and the "feeling" of the testing team regarding the maturity and quality of the product. The test team uses two "standard" test design techniques (domain testing and boundary value analysis). Other techniques are not considered and evaluated with respect to their efficiency in the project's context. Thus, decisions on the test model, the design technique as well as on coverage criteria are made implicitly, without a thorough analysis of alternatives.

Logical test cases and test data are explicitly defined on the basis of requirements and documented in a test management tool. Decisions concerning concrete test cases and test data are made explicitly and are mostly documented during the test execution in test protocols. The decision on the concrete test order is made explicitly, but only documented in the case of a failed test run. A matrix of concrete test environments is also managed by the testing team. Decisions on logical test environments as well as on the logical test order are made implicitly and are not documented.

The evaluation of a test run is made explicitly for each executed test case. In the case of a failure a process concerning the life cycle of a defect is passed, from its classification, localization and correction to its retest. At the end of a test cycle, the test team evaluates the results. This decision is made explicitly, but only summarizes the test results. Since the definition of the test end criteria is not performed, the evaluation of the test cycle occurs without reference to the defined criteria.

Implications: The decision-based analysis highlights the following main strengths and weaknesses of the testing process. Missing involvement of the testing team in decisions at specification level leads to input which is not well suited to be used in the testing process. Thus, complex user scenarios are not part of the documentation provided by requirements engineers. However, these scenarios would be very precious for the system testing as they lead to realistic test cases.

Another weakness concerns the unstructured decision process on the test goal as well as on the test strategy level. Thus, a thorough evaluation against goals is not possible. A main opportunity improvement must include methodologies which help testers to define objective and measurable goals in advance. A strength of the testing process is the thorough documentation of decisions concerning test cases and test data supporting the repeatability of test runs e.g. in regression testing.

4.4 Evaluation Framework for Testing Approaches in the Literature

The decision hierarchy can also be used as a framework for the comparison of testing approaches. It allows the classification of approaches depending on whether they provide (automated) support for a specific decision or not. Figure 4 exemplifies how approaches to use case based testing can be compared on the basis of the decision hierarchy. Comparing the approaches on the basis of the decision hierarchy allows the analysis of their commonalities and differences. As illustrated in Figure 4, some decisions, e.g. the decision concerning the test model, are supported by all approaches, whereas other decisions, e.g. the decision concerning quality requirements, are partially supported by only one subset of the approaches.

5 Conclusion

In this research work, we presented a generic decision hierarchy which contains decisions to be made during the testing process at different decision levels. We evaluated our hierarchy in four case studies.

Our decision hierarchy proved of value for both, for industry as well as for research applications. Practitioners get a deeper understanding of the complex decision making process during testing. Thus, the hierarchy can be used as an introducing guideline to the complex area of testing processes. Additionally, this approach increases the awareness of all decisions which have to be made during the testing process. The decision hierarchy is useful for researchers, too. First of all it enriches the body of knowledge on the subject of decision-making in the area of testing and builds the foundation for further research in the area of rationale management. Rationale management research aims at making design and development decisions explicit to all stakeholders involved. Additionally, as illustrated in the case studies, the decision hierarchy can be used by researchers as an evaluation framework in many contexts.

Based on our experience from applying this hierarchy in the case studies, we revealed that our approach is universal enough to be applied in different contexts. However, it is also specific enough to highlight the similarities and differences of the subject matters. Additionally, our approach is easily to be learned. Thus, students as well as practitioners get familiar with key issues of the testing process without having to go into details. Finally, our hierarchy eases the communication among testers by providing a common terminology.

	Decisions		Approaches								
Decision level	Generic tesing process	System testing process	Path analysis [1]	Extended UCs [4]	TOTEM [5]	Structural Testing with UCs [7]	ASM Based Testing [8]	Requirements by Contracts [18]	Testing with UCs [20]	SCENT [21]	Simulation and Test Models [23]
Specification level	Test basis	*Functional requirements*	X	(X)	(X)	(X)	X	(X)	X	X	X
		Quality requirements							(X)	(X)	(X)
Test goal level	Test focus	*Critical functional requirements*	X	(X)		(X)		(X)			
		Critical quality requirements								(X)	
	Test intensity	Test intensity	(X)								
	Test end criteria	Test end criteria									
Test strategy level	Test model	Test model	X	X	X	X	X	X	X	X	X
	Coverage criteria	*Model coverage*	X	X	X	X	X	X	X	X	X
	Test design technique	*Degree of automation*			X		X	X		(X)	X
	Ideal test order	Ideal test order			(X)						
Test Design Level	Logical Test Case	Logical Test Cases	X	X	X			X			
	Logical Test Environment	*External Systems*									
		Logical Automation Tools			X			X			
	Logical Test Data	Logical Test Data		X	X			X			
	Logical Test Order	*Test Case Order*									
Test Realization Level	Concrete Test Case	*GUI Steps*	(x)							(x)	
	Concrete Test Environment	Concrete External Systems, Concrete Automation Tools									
	Concrete Test Data	*GUI Data*	(x)							(x)	
		GUI Layout	(x)							(x)	
	Concrete Test Order	*Concrete Test Case Order*									
Test Run Level	Test Run Evaluation	Test Run Evaluation	X	X	X			X			
Test Evaluation Level	Test Cycle Evaluation	Test Cylcle Evaluation									

Figure 4. Application of the decision hierarchy to compare testing approaches, X = Approach supports decision, (X) = Approach partially supports decision

References

[1] N. Ahlowalia, “Testing from Use Cases Using Path Analysis Technique”, International Conference On Software Testing Analysis & Review, 2002.

[2] A. Aurum, C. Wohlin, A. Porter, „Aligning Software Project Decisions: a Case Study“. International Journal of Software Engineering and Knowledge Management, vol. 16, number 6, pp. 795 – 718, december 2006;

[3] B. Beizer “Software Testing Techniques”, Second Edition, Van Nostrand Reinhold, New York, 1990

[4] R. Binder “Testing Object-Oriented systems”, Addison-Wesley, 2000

[5] L. Briand, Y. Labiche, “A UML-based Approach to System Testing”, Technical Report, Carleton University, 2002.

[6] I. Burnstein, T. Suwannasart, and C. R Carlson, “Developing a Testing Maturity Model for Software Test Process Evaluation and Improvement”, Proceedings of the IEEE International Test Conference on Test and Design Validity, 1996

[7] A. Carniello, M. Jino, M. Lordello, “Structural Testing with Use Cases”, WER04 - Workshop em Engenharia de Requisitos, Tandil, Argentina, 2004.

[8] W. Grieskamp, M. Lepper, W. Schulte, N. Tillmann, “Testable Use Cases in the Abstract State Machine Language”, Second Asia-Pacific Conference on Quality Software, 2001.

[9] A. H. Dutoit, R. McCall, I. Mistrik, and B. Paech, “Rationale Management in Software Engineering”. Springer-Verlag Berlin Heidelberg, 2006.

[10] IEEE Std. 829-1998, Software Engineering Technical Committee of the IEEE Computer Society, IEEE standard for software test documentation, USA, 1998.

[11] IEEE Std. 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology. New York, September 1990

[12] International Software Testing Qualifications Board, ISTQB Standard Glossary of Terms used in Software Testing V1.1, (2005)

[13] T. Illes, B. Paech, “An Analysis of Use Case Based Testing Approaches Based on a Defect Taxonomy”. In IFIP International Federation for Information Processing, vol. 227, Software Engineering Techniques: Design for Quality, ed. K. Sacha, Boston Springer, pp. 211-222, 2006.

[14] T. Illes, H. Pohlmann; T. Roßner, A. Schlatter, M. Winter: „Software-Testmanagement Planung, Design, Durchführung und Auswertung von Tests - Methodenbericht und Analyse unterstützender Werkzeuge“, Heise Zeitschriften Verlag, 2006

[15] T. Koomen, and M. Pol, “Test Process Improvement: A step-by-step guide to structured testing”. Addison-Wesley, 1999

[16] G.J. Meyers, “The Art of Software Testing”, John Wiley & Sons, New York, 1979

[17] D. J. Mosley, and B. A. Posey, “Just Enough Software Test Automation”, Prentice Hall, July 2002

[18] C. Nebut, F. Fleurey, Y. Le Traon, J.-M. Jézéquel, “Requirements by contracts allow automated system testing”, Proc. of the 14th. IEEE International Symposium on Software Reliability Engineering (ISSRE'03), 2003.

[19] B. Paech, and K. Kohler, “Task-driven Requirements in object-oriented development”. In Leite, J. and Doorn, J. Perspectives on Requirements Engineering (2003). Kluwer Academic Publishers 2003

[20] C. Rupp, S. Queins, „Vom Use-Case zum Test-Case“, OBJEKTspektrum, vol. 4., 2003.

[21] J. Ryser, M. Glinz, “SCENT: A Method Employing Scenarios to Systematically Derive Test Cases for System Test”, Technical Report, University of Zürich, 2003.

[22] A. Spillner, T. Linz, and H. Schaefer, “Software Testing Foundations - A Study Guide for the Certified Tester Exam - Foundation Level - ISTQB compliant”. dpunkt.verlag, 2006

[23] J. Whittle, J. Chakraborty, I. Krueger, “Generating Simulation and Test Models from Scenarios”, 3rd World Congress for Software Quality, 2005.

[24] T. Wolf, and A. H. Dutoit, “Sysiphus: Combining system modeling with collaboration and rationale”, In http://wwwbruegge.in.tum.de/publications/includes/pub/wolf2004GIRE/wolf2004GIRE.pdf (2004)

[25] Salomé TMF, https://wiki.objectweb.org/salome-tmf/, last visited May 2007.

[26] Object Web Consortium, Open Source Middleware Konsortium http://www.objectweb.org/index.html, last visited May 2007.

[27] Bugzilla, http://www.bugzilla.org/, last visited May 2007.

Acknowledgment

We would like to thank all interviewees for their cooperation and help in providing information and insight into documents. Furthermore, we would like to thank Andrea Herrmann for her helpful comments and Doris Keidel-Müller for reviewing previous versions of this paper.

Dokument Information

Titel	The Testing Process - A Decision Based Approach
Datum	04.09.2007
Version	1.0
Status	Final
Verteilung	http://www-swe.informatik.uniheidelberg.de/research/publications/reports.htm

Document Information

Title	The Testing Process - A Decision Based Approach 20
Date	04.09.2007
Version	1.0
Status	Draft
Distribution	http://www-swe.informatik.uniheidelberg.de/research/publications/reports.htm