

Abstract

There are several approaches to risk-based testing. They have in common that risk is the focus when the tester chooses what to test. In this thesis we will combine some of these approaches and present our method for risk-based testing. The method analyses the risk for each part of the system and use a hazard analysis to indicate what can go wrong. The test efficiency and risk determine the tests priority. We have shown how a software tool can support our method and implemented a proof of concept. The implementation is presented and been tried out by an experienced tester.

Preface

This report is the result of my master thesis in Software Engineering at Department of Computer and Information Science, Norwegian University of Science and Technology (NTNU). The report discusses risk-based testing and the implementation of software tool to support risk-based testing is described.

I would like to thank my supervisor, Tor Stålhane, at the Department of Computer and Information Science, NTNU, for valuable help and feedback during the semester

Trondheim, June 20, 2005

Lars Kristoffer Ulstein Jørgensen

Contents

1. Introduction.....	7
2. Risk-based Testing – A State of the Art.....	8
2.1 Risk.....	8
2.1.1 Risk Exposure.....	8
2.1.2 Handling risk.....	9
2.1.3 Risk in software engineering.....	10
2.2 Testing.....	11
2.2.1 Tests and test cases.....	11
2.2.2 Black box and white box testing.....	11
2.2.3 The testing process.....	12
2.2.4 Testing and risk.....	13
2.3 Risk-based testing using hazards found in risk analysis.....	13
2.3.1 Preliminary Hazard Analysis (PHA).....	13
2.3.2 Failure Modes and Effect Analysis (FMEA).....	14
2.3.3 Hazard and Operability Analysis (HazOp).....	14
2.3.4 Heuristic risk-based testing.....	16
2.3.5 Prevention of the hazards.....	19
2.3.7 Risk-based testing for e-business using FMEA.....	22
2.4 Prioritizing tests by using consequence and probability.....	24
2.4.1 Cost factors.....	24
2.4.2 Defect generators from the process.....	25
2.4.3 Defect generators in the code.....	26
2.4.4 A case-study in using risk-based testing.....	27
2.4.5 Prioritizing tests against deadlines.....	28
3. Our method for risk-based testing.....	30
3.1 Prioritizing hazards and creating related tests and barriers.....	30
3.2 Prioritizing the modules in the system.....	32
3.3 Prioritizing the tests.....	35
4. Requirement specification for the software tool.....	38
4.1 Functional requirements.....	38
4.1.1 Project.....	38
4.1.2 Risk category.....	38
4.1.3 Module.....	39
4.1.4 Hazard.....	39
4.1.5 Test.....	39
4.1.6 Barrier.....	40
4.2 Non-functional requirements.....	40
4.2.1 Quality.....	40
4.2.1 Technology.....	40
5. Presentation of the software tool.....	41
5.1 Implementation.....	41
5.1.1 Data model.....	42
5.2 Screen shots.....	43

5.2.1 Select project.....	44
5.2.2 Create a new Project	45
5.2.3 Select and weight risk categories.....	46
5.2.4 List the modules.....	47
5.2.5 Valuation of the modules.....	48
5.2.6 Create a new test.....	49
5.2.7 List the tests.....	50
5.2.8 List the hazards.....	51
5.2.9 List the barriers.....	52
5.3 Trying out the software tool.....	53
6. Conclusion	54
7. Further work.....	55
7.1 Improvements of the method	55
7.2 Extending the functionality of the software tool.....	55
7.3 Try out the software tool.....	56
References.....	57
Glossary	59

List of Tables

Table 2.1: A risk matrix.....	9
Table 2.2: A PHA table.....	14
Table 2.3: An FMEA table.....	14
Table 2.4: A HazOp table.	15
Table 2.5: Scoring of subsystems using.....	16
Table 2.6: A Generic Risk List.	17
Table 2.7: A Risk Catalogue for installation.	18
Table 2.8: A component risk matrix.	19
Table 2.9: Risk analysis and test effective scores of the Test Process Worksheet.	23
Table 2.10: Risk Exposure for the function “Close Account”.....	28
Table 2.11: An example calculating the risk.	29
Table 3.1: The information we need from the hazard analysis.....	30
Table 3.2: A risk matrix for the risk exposure.	31
Table 3.3: A table with the information obtained in the hazard analysis.	32
Table 3.4: Risk categories. The risk factors used in the risk analysis.....	32
Table 3.5: Calculation of the cost number.	33
Table 3.6: Calculation of the probability number.	34
Table 3.7: The process used to find the cost number and the probability number	34
Table 3.8: The Risk exposure is calculated.	35
Table 3.9: Risk matrix used in the final test priority.	36
Table 3.10: A prioritized list of tests.	36

List of Figures

Figure 2.1: The ALARP region.	20
Figure 2.2: Wrapping.	21
Figure 5.1: A UML class diagram showing the data model for the software tool.	42
Figure 5.2: Screen shot of Start.aspx.	44
Figure 5.3: Screen shot of NewProject.aspx.	45
Figure 5.4: Screen shot of RiskCategory.aspx.	46
Figure 5.5: Screen shot of Modules.aspx.	47
Figure 5.6: Screen shot of ValuateModules.aspx.	48
Figure 5.8: Screen shot of CreateTest.aspx.	49
Figure 5.7: Screen shot of Main.aspx.	50
Figure 5.9: Screen shot of Hazards.aspx.	51
Figure 5.10: Screen shot of Barrier.aspx.	52

1. Introduction

Testing is done late in the development process when there are not much money and time left. The managers are eager to release the product. Since testing is time consuming and often underestimated they put a lot of pressure on the testers [8]. The testers have to do their job, but they cannot test everything. They have to test what they think is most important. By identifying and prioritizing hazards and risk factors, they can make sure that the limited time and resources are used to test the most important things and keep the risk as low as possible. The aim of the method for risk-based testing described in this paper is to help them doing this. After a study of the state of the art of risk-based testing, we will introduce the method and describe the implementation of software tool to support it.

2. Risk-based Testing – A State of the Art

Considering risk in software testing is not new. *“Traditional testers have always used risk-based testing, but in an ad hoc fashion based on personal judgment”* [5]. There are several approaches to risk-based testing. They all have in common that risk is central when choosing what to test. We will first discuss risk and testing separately in section 2.1 and 2.2 before describing two ways of doing risk-based testing in section 2.3 and 2.4.

2.1 Risk

Everyone has an idea about risk. We are taking risks every day. To get a reward it is often necessary to take risks. Crossing a busy street, risking a traffic accident to buy a coffee are examples of taking a risk to get some kind of benefit. Betting on the lottery is another example of taking a risk. In the latter example the risk of success is much smaller than the first, but the benefit of taking the risk is also bigger. Some people are more willing to take risks, while others are risk averse. According to Wikipedia [34], *“Risk is the potential harm that may arise from some present process or from some future event”* Risk is not yet a problem, but it can be if not the right actions are not taken.

2.1.1 Risk Exposure

When considering the risk we have to look at the consequence if the risk becomes a problem, and the probability that this will happen. The risk exposure is found by multiplying the consequence and probability of an event:

$$\text{Risk Exposure} = \text{Consequence} \times \text{Probability}$$

The consequence is the loss associated with the possible problem. Cost is often used instead of consequence to calculate the risk exposure. These are negative things like loss of time, quality, money, control or understanding. For example, big changes in the requirements for a system are likely to lead to losses of time and money if the design is not flexible enough [17]. To get an idea about the potential loss, the consequence has to be valuated.

How much it will cost in real money can be used for this purpose as we will see in section 2.1.2. It is often enough to give a score like low, medium and high. More detailed scores can be given, but it may lead to a lot of unnecessary discussions. Time should not be wasted on disagreement if, for example, the consequence is high or very high. The level of detail needed can be determined according to each situation.

The likelihood that an event can lead to harm is usually calculated as a probability. If expected loss in real money is used for the consequence in the calculation of the risk exposure, this will give the expected loss. As for the consequence, a score like low, medium and high may be as effective. The risk exposure can then be found using a risk

matrix as illustrated in table 2.1 taken from IEC 61508 [11]. In this matrix the risk is decreasing, I is intolerable risk. The risk is decreasing with higher number. IV is negligible. The risk exposure is highest in the upper left corner. This matrix is more detailed than the matrixes used later in this report.

<i>Probability</i>	<i>Consequence</i>			
	Catastrophic	Critical	Marginal	Negligible
Frequent	I	I	I	II
Probable	I	I	II	III
Occasional	I	II	III	III
Remote	II	III	III	IV
Improbable	III	III	IV	IV
Incredible	IV	IV	IV	IV

Table 2.1: A risk matrix.

2.1.2 Handling risk

After identifying the risks we have to consider what to do with them. It is important to look at the degree to which we can change the outcome before the risk becomes a problem. How can we minimize or avoid the impact of a risk? Risks can be prevented, transferred or accepted [16].

Prevention is a way to reduce the cost or the probability. This can be done by changing the conditions so the event will not happen. In software engineering, conducting more testing is a way to reduce the risk. It will reduce the uncertainty and may find critical faults which can then be corrected. To build barriers to protect the critical parts of the program if there is an error is another way to prevent risks. Barriers are discussed in section 2.3.4.

Transferring the cost means that the cost is put on another party. This can be the customer, a subcontractor or an insurance company. To transferring the cost to the customer is in itself a risk, since it can give you a bad reputation which again can lead to loss of customers and market share.

If the risk is accepted, two things should be done. First, the risk can be mitigated. This means that steps are taken to reduce the probability or the consequence of the risk. Second, a contingency plan should be defined. It explains what to do if the risk becomes a problem [16].

If the cost in real money and a probability value are used to calculate the risk exposure, it will give the expected loss. A risk cost of \$ 50000 with a probability of 0.02, will give a risk exposure of \$ 1000. The cost of prevention has to be considered against this value. If there are many risks, the sum of all risk exposures gives the expected loss, and can be used in the budgets. Such calculations are, however, more useful in finance than in computer science.

2.1.3 Risk in software engineering

Traditionally, risks are ignored in software engineering. An optimistic scenario is assumed and there are no funds reserved for dealing with risks [31]. But things do not always go as planned, and in every software development project there are a lot of risks that need to be considered. Since there are always many uncertainties in software engineering, how the development team handles risks will have strong impact on the success of the project. To identify the risk is an important part of new development processes like Rational Unified Processes (RUP) [16].

The risks related to the product are usually analysed separately from the risk related to the project process [35]. The product risk can be found by identifying potential hazards and threats to the system. Hazard is a potential unwanted event. The term used in evaluating safety. The seriousness of the hazards will depend on the nature of the software. In safety-critical systems the consequence of hazards will normally be much higher than in commercial software, but the term “hazard” can be used in both places.

Software with many critical failures will not only cause problems now, but may also affect the business in the future after the failures have been fixed. If the customer encounters problems and feels that his time has been wasted, the customer will not come back [6]. Hence a good user interface is also a critical risk factor. This will especially be relevant for e-business.

For commercial software, the most serious hazard may be the loss of data. For safety critical software the consequence of a failure can be much bigger. A hazard can lead to disasters and many human lives can get lost. In safety and business critical systems the importance of controlling the risks is high because of the consequence of failure. In these kinds of systems, reliability is very important [35]. But highly reliable software does not necessarily ensure a safe operation. It must also be trustworthy [35]. When software fails we must prevent dangerous behaviour. This is described further in section 2.3.3.

The process risk is related to how the process of developing the software has been. Process risk linked to a flawed development process can result in introduction of faults that can lead to reduced reliability and reduced safety [35]. Therefore, both the product risk and the process risk should be considered. The risk analysis methods described in section 2.3 are considering the product risk whereas section 2.4 is focusing on the risk factors related to the process.

2.2 Testing

Testing in software engineering *“is a process used to identify the correctness, completeness and quality of developed computer software. Actually, testing can never establish the correctness of computer software, as this can only be done by formal verification (and only when there is no mistake in the formal verification process). It can only find defects, not prove that there are none.”* [34].

The purpose of testing is to find faults in the system and build confidence that the system can provide the functionality specified in the requirements and not cause losses to the customer [26]. Both of these are important. Gerrard [6] adds two more objectives, *“to provide evidence that business benefits required from systems are available”* and *“to provide data on the risk of release (and use) of the system under test.”* [6]

2.2.1 Tests and test cases

There is not a common definition of test case. IEEE defines a test as a set of one or more test cases [10]. A test case is *“a condition to be tested that includes its own identification and the expected response”* [9]. Kaner [14] describes a test case as a question you ask the program to gain information. Furthermore, a test case may not be too specific, more like an idea on what to check in the program. The important thing is that the test cases are capable of revealing information. They are not necessarily designed to expose a defect. I will define a test case as a set of instructions designed to detect a particular class of defect, by bringing about a failure [18]. A test case consists of test item, input and output specification and environmental needs. In the rest of the report we will use the term tests and not test cases.

2.2.2 Black box and white box testing

Most risk-based testing is black box testing. This technique is also called functional testing or specification-based testing and it checks the system against the requirement specification. It does not look at the internal structure, but provide the system with inputs and observe the outputs. Black box testing *checks “that the outputs of a program, given certain inputs, conform to the functional specification of the program. The term black box indicates that the internal implementation of the program being executed is not examined by the tester. For this reason black box testing is not normally carried out by the programmer. In most real-world engineering firms, one group does design work while a separate group does the testing”* [34].

Risk-based testing should be used together with other techniques. Inspection and continuous code testing during implementation are also important. This is called white box testing and it checks *“that the outputs of a program, given certain inputs, conform to the internal design and implementation of the program. The term white box indicates that the tester closely examines the internal implementation of the program being tested”* [34]. Often, most of the errors are found during this earlier phases of testing [23] and are

easy to correct. Black box testing, however, is also essential to ensure the quality of the system before it is delivered.

2.2.3 The testing process

We have discussed two ways of doing testing; black box and white box testing. How testing is performed depends on the development process and the nature of the software developed.

According to Stålhane [26], it is important to think of testing from the beginning of the project. *“As soon as anybody states a requirement, somebody must be responsible for making a test for it. A requirement without a test will be ignored and forgotten”* [26]. This is also important when using RUP, where testing is supposed to start as early as possible [8].

As stated in section 2.2.1, testing consumes a high percentage of system development costs, and it is thus important to plan the testing process to make it efficient. Testing is performed in all phases of the development process, but the activities change. Early stages are done by programmers and consist of inspections and unit testing.

Hutcheson [9] describes two approaches to the testing process, bottom-up and top-down. The bottom-up approach combines a few modules after unit testing and these are tested until they are stable. Then a few more modules are added and the system is tested again. This is done until all the modules are added. The bottom-up approach is slow. As Hutcheson [9] states, it has a major drawback: *“Testers are testing the simulators, not the real system. I haven’t seen a real system yet that behaved exactly like the simulator”* [9].

Hutcheson [9] recommends the top-down approach where as much of the system is put together as soon as possible. If possible the system is then tested from the user interface. The top-down approach begins with the integration phases. This demands that each module is working and thoroughly unit tested. The system is then tested as a whole. The methods described in this report will work best for the top-down approach.

2.2.4 Testing and risk

Testing is often done late in the project when there is not much time or money left. Short time to market and reducing costs are important, and there will therefore never be enough resources. If the system should be delivered in time, a lot of things will not be tested. Managers will often choose to deliver the system on time rather than deliver a system that works perfectly [6]. Testing is expensive and often underestimated. *“In many organizations, software testing accounts for 30 to 50 percent of software development costs. Yet most people believe that software is not well tested before it is delivered”* [8]. If the estimates were realistic, it would take even more resources away from the rest of the process. The result would be that they fail to deliver enough functionality.

The complexity and size of software products makes it practically impossible to develop a system without faults. Myers [19] shows an example of a program consisting of only 20 lines of code, in most programming languages will have more than a trillion different states to test. To test all the states is an impossible task for a tester. When it is so difficult to test a simple program, then commercial applications with hundred of thousands lines of code will have a lot of uncertainty. There will always be faults never identified and this will represent a potential risk. On the other hand, many of these faults will never occur. Adam [1] published some data demonstrating that only a small percentage of the faults caused most of the problems. To find these faults is more important than finding the others.

All this puts a lot of pressure on the testers. By identifying and prioritizing hazards and risk factors, we can make sure that the limited time and resources are used to test so that we to keep the risk as low as possible. In the next section we will describe several ways of doing this.

2.3 Risk-based testing using hazards found in risk analysis

A risk analysis will try to identify and assess the factors that may jeopardize the safety of the system [23]. We include a short introduction to some ways of doing risk analysis, Preliminary Hazard Analysis (PHA), Fail Mode Effect Analysis (FMEA), Hazard and Operability Analysis (HazOp) and heuristic analysis. We continue by looking at how the hazards can be prevented. Section 2.3.7 describes how the hazards found can be used to prioritize tests.

2.3.1 Preliminary Hazard Analysis (PHA)

All risk analysis starts by identifying hazards [20]. This should be done systematically. PHA is normally done early in the design phase. It tries to identify potential hazards early in the project so that they can be eliminated, minimized or controlled before it is too late [20]. For each hazard there may be many causes and effects, and for each cause there may be more than one preventive action. An advantage of PHA is that it does not need

much information to be carried out. Table 2.2 shows an example of a PHA table. It is taken from Stålhane [25].

<i>Hazard</i>	<i>Cause</i>	<i>Main effect</i>	<i>Preventive action</i>
Wrong diagnosis retrieved	Wrong diagnosis inserted	Kill or hurt the patient	Double check all patient info inserted

Table 2.2: A PHA table.

2.3.2 Failure Modes and Effect Analysis (FMEA)

FMEA is usually done in the construction face. The intention is to identify the parts of the system that will need improvements to meet the safety and reliability requirements [20]. FMEA is mostly a qualitative analysis, except for the seriousness that may be given in a probability or frequency number. The first step is to find how a unit can fail and find the effect. The second step is to consider the severity of the failure mode. FMEA is an easy method to use, but it needs a lot of knowledge about the system. Another problem is that it only works if the failures are not dependent on each other.

<i>Unit</i>	<i>Failure mode</i>	<i>Effect of failure</i>	<i>Failure rate</i>	<i>Severity</i>	<i>Action</i>

Table 2.3: An FMEA table.

2.3.3 Hazard and Operability Analysis (HazOp)

HazOp is a more formal and systematic way of doing risk analysis [20]. It will need the system's structural description (architecture) to define the study nodes. This means that HazOp must be carried out later in the development process, when the architecture is well known. The study nodes are the points of the system where we focus the analysis. This can be points where the system interacts with its environment (input, output), or where parts of the system exchange information.

HazOp contains two structuring devices, the table and the guide words. An example of a HazOp table taken from Stålhane [25] is shown in table 2.4. The guide words explain something about the study node. Traditionally words like: “none”, “less”, “more”, “part of”, “complementary”, “else than”, are used [20]. These were originally used in the

process industry. All of these guide words are not appropriate in software systems. One solution is to create new guide words. Another is that the participants of the HazOp can agree on what they mean in this situation. Studies have showed that the latter approach has been the most effective. The guide words focus the attention of the participants, and they can ask questions like: “What does the guide word mean in our system or for this study node?”

<i>Guide word</i>	<i>Study node</i>	<i>Consequences</i>	<i>Causes</i>	<i>Possible solutions</i>
Less	Patient info	Incomplete info Wrong treatment Kill or hurt patient	Missing updates Lost updates Incomplete updates	Check and sign-off for updates Mirror database

Table 2.4: A HazOp table.

Stålhane [28] describes how to use HazOp to find the subsystems with most hazards. UML use cases are used as a starting point. Use cases have no study nodes; thus a standard HazOp with guide words could not be used. Instead, a functional HazOp is performed based on the functionality offered by the system. The procedure has the following steps:

- Step 1: Prepare use cases for the subsystem to be analysed.
- Step 2: A warm up exercise looking at previous risk analysis.
- Step 3: Perform function-based HazOp
 - How can this function fail?
 - What are the consequences for the stakeholders, the service receiver, the service provider and the development company? This will give a list of hazards
- Step 4: Document result and obtain feedback from the participants
- Step 5: Assess severity of each hazard

A score from zero to three is used to indicate the severity given to each hazard, where three is the most serious. Different stakeholders may assign different severities to the same hazard. For each subsystem and stakeholder group, the number of functions that receive each hazard value is registered. The score for each function can be found in two ways, the weighted average and the score to the majority of the functions. In the latter case, the median is used if the majority criterion did not give a unique result. Table 2.5 shows an example of calculation of scores for the subsystems, taken from Stålhane [28].

Subsystem	Seriousness				Score	
	3	2	1	0	Average	Max.
D1	1	-	1	-	2.0	2
B1	-	-	3	2	0.6	1
O1	-	-	3	1	0.8	1
X1	1	3	2	1	1.6	2
I1	6	-	3	-	2.3	3
G1	2	1	1	-	2.3	3
M1	-	1	2	-	1.3	1
A2	1	1	2	-	1.8	1

Table 2.5: Scoring of subsystems using.

2.3.4 Heuristic risk-based testing

Bach [4] describes a heuristic analysis to do risk-based testing. “A heuristic is a way of directing your attention fruitfully” [34]. The method used by Bach is not final or strict. Human experience is used to find the most risky areas. Bach presents two approaches to his heuristic risk-based testing, inside-out and outside-in.

In the inside-out approach, the tester studies the product together with the developer and asks questions like “What can go wrong here?” For each part of the product the tester will look for vulnerabilities, threats and victims. This is the same approach we used in the FMEA.

The Outside-in approach begins with a set of potential risks and matches them to the details of the situation. A list of predefined risks is consulted and used to determine whether they apply here and now. Bach uses three kinds of lists: quality criteria categories, generic risk list and risk catalogues.

Quality Criteria Categories is designed to evoke requirements such as: capability, reliability, usability, performance, installability, compatibility, supportability, testability, maintainability, portability, and localizability

Generic Risk List is a list of risks that are applicable to any system:

- *Complex*: anything disproportionately large, intricate, or convoluted.
- *New*: anything that has no history in the product.
- *Changed*: anything that has been tampered with or “improved”.
- *Upstream Dependency*: anything whose failure will cause cascading failure in the rest of the system.
- *Downstream Dependency*: anything that is especially sensitive to failures in the rest of the system.
- *Critical*: anything whose failure could cause substantial damage.
- *Precise*: anything that must meet its requirements exactly.
- *Popular*: anything that will be used a lot.
- *Strategic*: anything that has special importance to your business, such as a feature that sets you apart from the competition.
- *Third-party*: anything used in the product, but developed outside the project.
- *Distributed*: anything spread out in time or space, yet whose elements must work together.
- *Buggy*: anything known to have a lot of problems.
- *Recent failure*: anything with a recent history of failure.

Table 2.6: A Generic Risk List.

Risk catalogues is a list of risks that belong to a particular domain. Below follows an example from installation. Problems that may occur in this domain are listed.

- Wrong files installed
 - o temporary files not cleaned up
 - o old files not cleaned up after upgrade
 - o unneeded file installed
 - o needed file not installed
 - o correct file installed in the wrong place
- Files clobbered
 - o older file replaces newer file
 - o user data file clobbered during upgrade
- Other apps clobbered
 - o file shared with another product is modified
 - o file belonging to another product is deleted
- HW not properly configured
 - o HW clobbered for other apps
 - o HW not set for installed apps
- Screen saver disrupts install
- No detection of incompatible apps
 - o apps currently executing
 - o apps currently installed
- Installer silently replaces or modifies critical files or parameters
- Install process is too slow
- Install process requires constant user monitoring
- Install process is confusing
 - o UI is unorthodox
 - o UI is easily misused
 - o Messages and instructions are confusing

Table 2.7: A Risk Catalogue for installation.

The three kinds of lists described above can be used to:

1. Decide what component or function to be analysed
2. Determine the scale of concern. Everything is assumed to have a normal risk unless there are reasons to believe something else.
3. Gather information about the things you want to analyse more
4. Determine each risk's importance
5. Record any other risk that are not on the list
6. Record any unknowns which impact the ability to analyse the risk
7. Check the risk distribution

Three ways to organize risk-based testing is suggested, risk watch list, risk/task matrix and component risk matrix. **The risk watch** list is a list of risks that you periodically review during the project to be aware of the most common risks.. **Risk/task matrix** sorts

the risks according to their importance. The tasks to be invested in to minimize the risk are listed for each risk. This is a tool that is most useful when resources for testing are negotiated. The **component risk matrix** breaks the product into 30 or 40 areas or components. In the left column the components are listed. In the middle the scale of concern, “low”, “normal” or “high” risk is listed. In the right column the risk heuristics for that component are listed. The risk heuristic indicates the risk for that component. During testing, the components are tested according to their risks as specified in the matrix.

Component	Risk	Risk Heuristics
Printing	Normal	Distributed, popular
Report Generation	Higher	New, strategic, third-party, complex, critical
Installation	Lower	Popular, usability, changed
Clipart Library	Lower	Complex

Table 2.8: A component risk matrix.

2.3.5 Prevention of the hazards

To prevent the identified hazards from becoming real problems, they will need extra attention. Prevention of failures is important also for testers as Gerrard states. *"Testers are responsible for preventing, as well as detecting faults"* [6]. They must get the developers to take actions to reduce the risk.

IEC 61508 [11] defines a few terms related to risk reduction. It points out that reducing the risk when implementing safety-related systems is important. To reduce the risk so it meets the tolerable risk for a specific situation is called necessary risk reduction. The tolerable risk is dependent of the frequency (probability) and the consequence of the specific hazardous event. A safety-related system contributes to meet the necessary risk reduction. The method to achieve tolerable risk is called *"as low as reasonable practicable"* (ALARP). The level of risk is divided into regions. The region between intolerable risk and acceptable risk is called the ALARP region. This is defined as a region where there is still a lot of risk, but it can be accepted if a benefit is desired. The spending on reducing the risk will depend on how big the risk is [11]. This is illustrated in figure 2.1 showing the ALARP region in the middle.

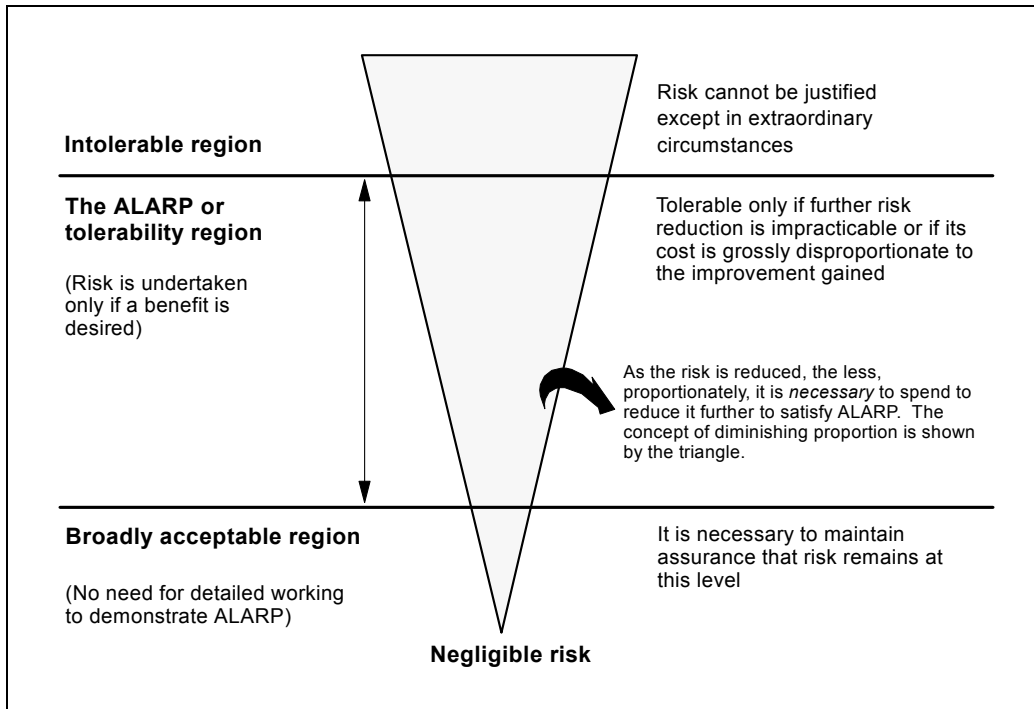


Figure 2.1: The ALARP region.

To reduce the risk, the most serious hazard should be given special focus. Focusing on testing and analysis of the code may be one solution. Another opportunity is to analyse the structures of the program and minimize the risk. A way to do this is to use fault tree analysis (FTA). This analysis can be useful in many domains, but is not the most efficient method in software. This is especially so in safety critical systems where it is better to ensure safety by using redundant and fault tolerant systems rather than expanding design capacity [9]. In complex software, it is difficult to be sure that everything is right and it is time consuming [15]. Thus, we must avoid too complex solutions.

An easier and more effective technique is to prevent the event causing the hazard to take place, or to prevent or reduce the impact of this event [3]. This is called a barrier. Smith [22] defines barriers as a *"hindrance that may either (i) prevent an action from being carried out or an event taking place, or (ii) prevent or lessen the impact of the consequences, limiting the reach of the consequences or weakening them in some way"* [22]. This may also be a socio-technical solution such as non-smoking sign or a warning.

Stålhane [24] explains how wrappers can prevent COTS-components in a software system from doing harmful operations. Three different ways of construction wrappers are presented, isolation, wrapping and object programming and inheritance. The methods are illustrated in figure 2.2 below. The figure is taken from Stålhane [24].

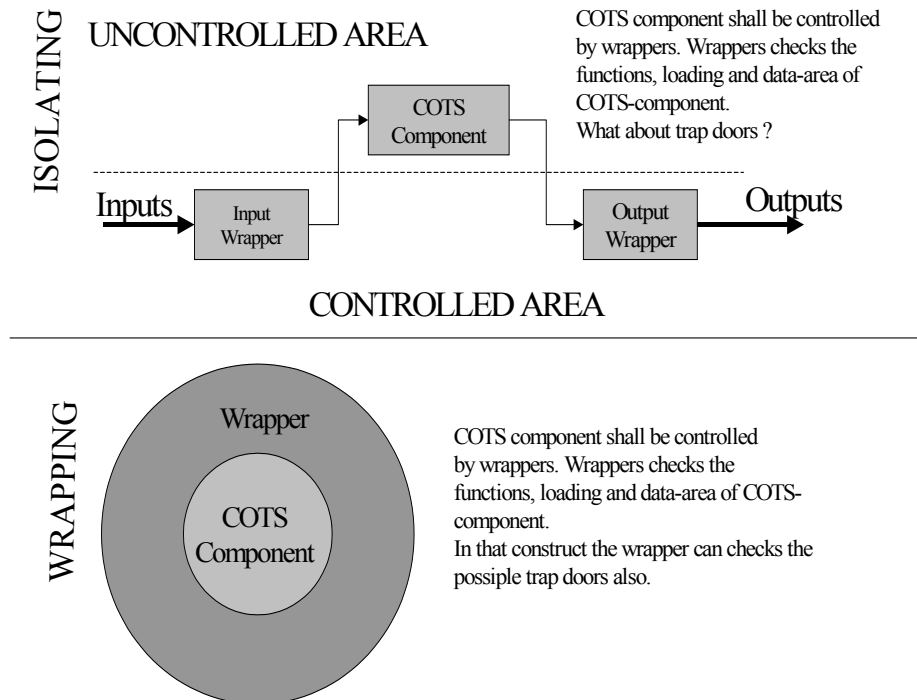


Figure 2.2: Wrapping.

Anderson [3] introduces a case study where an Off-The-Shelf component is used as a controller for a simulated boiler system. The use of barriers to detect errors and unwanted behaviour is demonstrated. Error symptoms and the necessary corresponding actions are identified. The errors are only checked in the interface between these two modules. This did not identify the faults were the errors where, but protected the interface.

Tan [30] shows an example on how to implement wrappers as interfaces and wrapper classes in Java. This means that the original class and the wrapper class implement the same interface. This makes it easy to choose to use the wrapper class. Wrappers are a kind of barriers and we will in this report use the term barrier for wrappers. We will also talk about barriers as small programs protecting modules or classes from hazards.

Smith [22] uses examples from computer-aided mammography and airspace route feasibility to demonstrate barriers. In each of these case studies, a risk analysis was performed to identify hazards. Most of the barriers were preventive and human oriented, e.g. staff training. Preventive barriers do not have any fault tolerance such as protective

barriers do. The lack of protective barriers in these case studies will lead to potential hazardous system. The case studies also showed that there are normally not a one-to-one relation between hazards and barriers.

Barriers are often supposed to protect against states and values that are not allowed and are difficult to test. We have to get the system into the required state for the hazard. Then the input that will cause the hazard has to be used. The systems response has to be observed and documented. To get the system into the required state often has to be done by force. A way to do this is fault injection [32]. Fault injection is described in Whittaker [33]. Ghosh [7] describes several case studies where fault injection is used to test software survivability.

Implementing barriers is an effective way to protect against hazards and reducing the risk. There are still some problems with barriers. They are not treating the real problems where the errors are, but only treating the symptoms [32]. The barriers may prevent the hazard, but the program will still contain errors and cause problems. We have discussed how to test the barriers with fault injection. Barriers are extra code lines that add extra complexity. They can also be complex and defective themselves [32]. The tests should verify that the barrier prevents the hazard from occurring, prevents the hazard from causing a safety-critical error and that the critical condition cannot occur.

2.3.7 Risk-based testing for e-business using FMEA

Gerrard [6] uses a risk-based approach to test e-business, but his method can be applied to other kinds of system as well. The process consists of five stages: risk identification, risk analysis, test scoping and test process definition. A table called Test Process Worksheet is the main working document in the method and it is completed in a four stage process. Each stage is discussed below. The rows in the Test Process Worksheet consist of a failure mode, also called a risk. The columns consist of scoring and prioritisation, assignment of test objective, effort, costs and so on for this failure mode. The test objective is the purpose of some test usually to address a specific failure mode.

Stage 1: Risk identification

An inventory of potential failure modes is prepared. These are derived from checklists. When the failure modes are identified, complex failure modes are split up and duplicates are discharged. The checklists are similar to the checklists that Bach [4] is using, described in section 2.3.2.

Stage 2: Risk analysis

In this stage, a risk workshop is convened with representatives from the business, development, technical support and testers. Each risk is considered, and the probability and the consequence are assessed. The risk exposure is calculated. Table 2.5 shows an example.

Stage 3: Risk response

If the risk is testable, it is turned into a test objective using the risk description. An example can be for the failure mode: “HTML used may work on one browser, but not others” [6] can have the test objective “Verify that HTML is compatible across supported browsers” [6]. A risk effectiveness score between one and five that relates to how confident the tester is that the risk could be addressed through testing is given to each risk. A value of one indicates that the tester does not have any confidence. A score of five is given if the tester have high “confident that testing will find faults and provide evidence that the risk has been addressed” [6]. The product of the risk exposure and test effectiveness will give the test priority number.

ID	Risk	Probability	Consequence	Risk Exposure	Test Effectiveness	Test Priority Number
Client Platform						
04	Invalid or incorrect HTML may exist on pages without being reported by browsers or visible to developers or testers.	4	4	16	5	80
05	HTML used may work on one browser, but not others	4	4	16	4	64
06	Users with client machines set up to use foreign character sets may experience unusual effects on their browsers. Foreign characters may be rejected by the database.	3	4	12	4	48
Component Functionality						
13	Spelling mistakes on Web pages may irritate users and damage our credibility.	5	4	20	5	100
Integration						
19	Links to on- or off-site pages that do not work make the site unusable	4	4	16	5	80

Table 2.9: Risk analysis and test effective scores of the Test Process Worksheet.

Stage 4: Test scoping

Review activity, need involvement from all stakeholders and staff with technical knowledge. The meeting discuss and agree on a total budget for testing. The Test Priority Number and the estimated cost to run the test are used to choose what test objectives to

include in the scope for the testing. The responsible for each testing object is agreed upon.

Stage 5: Test process definition

The scope of the testing is agreed upon and it is now possible to compile the test objectives, assumptions, dependencies, and estimates for each test stage and publish a definition for each stage in the test process. The stage-by-stage test process is documented.

2.4 Prioritizing tests by using consequence and probability

In the previous section the focus was on finding how software can fail to decide what to test. In this section we will look at factors that will make some parts of the system more risky than others. We will first look at factors that will affect the cost of failures. Parts of the system where these are important will have a higher consequence if something goes wrong. Factors increasing the probability related to the development process and the code is discussed in section 2.4.2 and 2.4.3. These factors are called defect generators, since they are used to identify parts of the system with many defects. Two examples of using some of these factors are described by Schaefer [21] and in the case study of Amland [2]. Since all of these factors can be used for our method described in chapter 3, we will explain the risk categories in more detail. We will also propose how some of them can be estimated.

2.4.1 Cost factors

There are several ways to find cost factors. Cost factors are factors considered for a part of a system that will cause bigger loss if there are failures in that part. Schaefer [5] looks at how critical, visible and used each part of the system is. Amland [2] has another perspective; he looks at the consequences the failures have for the vendor or customer. Consequences can be loss of reputation, higher maintenance cost or legal action. A combination of the two views will be used in our method to find the factors affecting cost. We will discuss both perspectives in this section. Some of the factors are related. A failure in a critical part of the system can lead to loss of reputation.

Failures in some parts of the systems may be more critical than other parts. To get an idea of how critical a failure will be, its consequences must be analysed. If the customer cannot work with the software and wastes time and money because of the failure, or if it leads to lost or corrupted data, it will give a high cost. If failures in that part of the system cause a user to use some workaround to achieve the goal, it is less critical and the cost can be considered medium. A part of the system is less critical if a failure is only hindering the user, thus causing workarounds or just making the program less appealing.

Some parts of the system are more visible and many users will experience failures if they are present in these parts. The forgivingness of the user should also be considered. Software for untrained or naive users like the general public needs careful attention to user interface and has to be more robust than functions only used by expert users [21].

How often a function is used is also important when the cost is considered. Some functions may be used every day, others only seldom. Some functionality is not visible from the outside, but is still often used, such as process control systems [21].

Failures in some parts of the system can cause loss of reputation for the customer or vendor. A loss of reputation can result in losing customers and market shares. More critical, used and visible parts of the system as discussed above will normally cause higher loss of reputation. Failures that catch the attention of by the media can be damaging for the company

Maintenance will on the average consumes 50-75% of the total cost of ownership [28]. About half of this goes to developing new functionality, but correcting faults also is a high cost. Parts of the system with many faults will be more expensive to maintain.

Legal consequences can be relevant in some systems. Violating the law can have big consequences for the companies affected, by fines or other forms of legal action. Legal consideration can be governmental restrictions on privacy and information. Examples of application of such a concern are systems where personal information is stored where failures can lead to legal problems for the company.

2.4.2 Defect generators from the process

The probability of failure is increased in parts of the system with more defects. We will look at several factors that can generate defects. Some of the defect generators are software metrics related to the code, like complexity, the number of faults found during inspection or earlier testing and changes during development. These are discussed in the next section. In this section we discuss defect generators from the process like introduction of new methods, tools or technology and how many and how qualified the developers that worked on each part of the system are. Defect generators taking the concentration away from fault removal activities like time pressure and code optimization are also discussed.

Developers using new methods, tools or technology may affect the number of faults. The developers may perform more errors in the beginning of the process, and fewer later when they learn how to use the new technology. There is always risky to be one of the first to use new methods, tools or technology because there are often problems there that have not yet been found.

The number of people involved can influence the code quality. It is normally better with a small group of highly qualified people, than bigger groups of less qualified people. The bigger groups of not so qualified people may seem to do as much for the same cost, but they are more likely to perform errors that may cause problems later. Parts of the system where less qualified people have been employed need more testing. Many companies employ their best people for the most complex parts. In this case parts of the system where less qualified people have been employed do not necessarily contain more faults.

Time pressure during development can influence how well the implementation is done. When there is not much time available, the programmers are more worried about getting the job done than about avoiding errors and often skip quality control activities. Working overtime may cause people to lose their concentration and they will do more errors when they write the code.

Some parts of a system need to keep the machine time and use of memory low. These parts must be optimized, which require extra design effort. In parts of the system with much optimization or time pressure, resources will be taken away from defect removal activities and this will result in more faults. As mentioned in section 2.2, most faults are found during inspections and early testing so taking shortcuts in these early testing phases may lead to high failure rates.

There may be other local factors for the project that is worth considering. Communication between people is important. This will suffer if the programmers are distributed geographically.

There may also be conflicts between managers that affect the number of faults. Important employees may have quit the job. This is a problem if they are the only one with knowledge important for that organization.

2.4.3 Defect generators in the code

Complexity is the most important defect generator, but is difficult to assess. The complexity predicts problematic parts of the system in the code and the likelihood for defects increases with higher complexity. About 200 different measures for complexity exist [21], but none of these are validated. We can do several complexity analyses, based on different aspects of complexity. The size can be used to indicate complexity. The size is normally measured by line of code (LOC) excluding comments and blank lines. It is commonly agreed that only executable LOC should be counted. Line of codes is language dependent and dependent on how the developer writes the code. Research has indicated that there may be an optimal program size that can lead to the lowest defect rate [13].

An example of a metric for cyclomatic complexity is McCabe's Number. This is calculated from the number of simple paths in the program. A guideline of the complexity is that a number between one and ten is a simple program, with low risk. A

McCabe's Number between eleven and twenty is more complex, medium risk. A program with a complexity higher than this has a high risk [13]. To find McCabe's number, the internal structure has to be known. This is a time consuming operation and unless it is not already done, it is not necessary to do it just to use this method. If there are no metrics available, the developers can give their personal estimate of the complexity. Most developers should be able to give a reasonable estimate on the complexity as low, normal or high. Experience shows that developers are good at making a qualitative assessment of quality attributes [29]. It can be an advantage if two developers assess it together to agree on an estimate.

Number of faults found during inspection or earlier testing phases can give an indication of where to expect many defects later. When the number of faults is used, the size should also be considered. An estimate for quality is the number of defects divided by the size of the code. The size can be LOC or functional points, if it is calculated [13]. Using defects found during "*inspection, the determination of a defect is based on the judgment of the inspectors. Therefore, it is important to have a clear definition of an inspection defect*" [13]. If there is an older version of the system that has been tested, we can also look at where there were many defects in that version and transfer the experience to the new system. It could also be helpful to study the testing done in earlier projects to analyse what the kind fault typically overlooked.

Parts of the system with many changes during development may have more defects than other parts. Changes often seem easy, but are often done under time pressure. They often cause problems because they are not analysed thoroughly. The changed parts of the system may have a bad design from start or the changes may have destroyed the original design. The number of changes done during the development process should be registered.

Amland [2] uses design quality as one of the important factors to indicate the probability of fault in his risk analysis. This could be measured counting the number of Change Requests to the design. The design quality depends on the function, and on the designers' experience of the design and the application.

2.4.4 A case-study in using risk-based testing

Amland [2] describes a case study in the development of a retail banking application. During the project, they realized that there were not enough resources to test everything. The system consisted of two parts and each used a different approach to decide what to test.

In the first part of the system, a combination of the top 20 most important transactions and all transactions with more than 10 faults from previous testing were tested. This was early in the project and very little information was available from the construction teams.

Later, when more information was available, more factors were considered. The cost of fault for customer $C(c)$ and vendor $C(v)$ in the areas of maintenance, legal issues, and reputation were considered for each function. The cost of fault for the vendor and customer were considered as equally important - thus the cost for each function was calculated as: $(C(c) + C(v)) / 2$

The probability of fault for each function $P(f)$ were estimated. This depended on whether the code was changed, if there was new functionality, the design quality, the size and the complexity. These factors were given weights from one to five, where a higher score indicating the poorer quality. Then, for each function of the system the probability for all indicators is considered, and given values from one to three. The weights and values for each function are multiplied. The probability *“is calculated as the Weighted Average of a weighted function divided by the highest Weighted Average of all functions”* [2]. This gave probability numbers between zero and one.

The risk exposure for each function, $RE(f)$, was calculated by multiplying the cost of fault and the probability using the formula:

$$RE(f) = P(f) * (C(c) + C(v)) / 2$$

The testing resources were focused on the functions with the highest risk exposures. The risk-based approach for testing used in this case study consumed less resources than the original estimate based on a traditional test approach.

	Cost			Probability						
	$C(v)$	$C(c)$	Average Cost	New	Quality	Size	Complexity	Weight average	Prob $P(f)$	$RE(f)$
Close Account	1	3	2	2	2	2	3	7,75	0,74	1,48

Table 2.10: Risk Exposure for the function “Close Account”.

2.4.5 Prioritizing tests against deadlines

Scheafer [21] focuses on prioritizing what to test, by finding the most important and worst parts of the product. The most important parts of the system are found using factors like cost of failure, most visible and most used parts of the product. The worst parts of the system are parts where the probability of defects is high. These parts are found using defect generators like complexity, changed areas, new technology, new solutions, new methods, new tools, number of people involved, where there was time pressure and local factors.

Weights are assigned for each relevant cost factor and defect generator. For each part of the system, values are assigned for the factors and the defect generators. Higher values mean that the area is more important or worse than others. This is illustrated in table 2.11. The table is taken from Schaefer [21]. These values are multiplied by the weights and added together. The highest values give the most risky parts and should be prioritized.

Area to test	Business criticality	Visibility	Complexity	Change frequency	RISK
Weight	3	10	3	3	
Order registration	2	4	5	1	46*18
Invoicing	4	5	4	2	62*18
Order statistics	2	1	3	3	16*18
Management reporting	2	1	2	4	16*18
Performance of order registration	5	4	0	1	55*3
Performance of statistics	1	1	0	0	13*0
Performance of invoicing	4	1	0	1	22*3

Table 2.11: An example calculating the risk.

3. Our method for risk-based testing

In the “State of the Art” chapter we discussed several methods for risk-based testing. In this chapter we will combine some of these in our method for risk-based testing. Hazards found in a risk analysis and the risk for each part of the system is used together to find and prioritize the tests. The method presented in this chapter is implemented in a software tool. The requirements are found in chapter 4 and an implementation of these is presented in chapter 5.

3.1 Prioritizing hazards and creating related tests and barriers

The first and possibly most important stage of our method for risk-based testing is to learn more about potential failure modes in the system we are going to test. Table 3.1 shows the information we need from the analysis.

Hazard	What can go wrong? (failure modes)
Cause	What action or states in the program will this hazard occur?
Module	What module or subsystem will this hazard occur?
Probability	How likely is it to happen, low, medium, high probability?
Consequence	What is the consequence, low, medium, high?
Risk exposure	Use a matrix or formula to calculate the risk from probability and consequence
Barrier	Can we and how can we implement a barrier?
Tests	How can we know that this will not happen?

Table 3.1: The information we need from the hazard analysis.

First we have to identify potential hazards in the system. Several kinds of hazard analysis were discussed in section 2.3. This method does not depend on one particular hazard analysis method. The choice of techniques depends on the project and previous experience of the participants. A combination of the methods can also be effective. It is important to identify hazards as early as possible, thus a PHA may be performed early in the project to find the most obvious hazards. A HazOp can be performed later when there is more knowledge about the system.

It is important to find the cause of the hazard. This will be the base when the barriers and tests are created. The cause may be a particular fault in the system, an abnormal behaviour from the user of the system or special environmental states of the system. Often a hazard will happen when more than one thing goes wrong at the same time [20].

The hazard will normally be related to one particular part of the system. Many faults in this module or subsystem will increase the likelihood for a hazard to become a problem. How to decide the risk exposure for a module is discussed in section 3.2.

The probability that the hazard will happen has to be estimated. We prefer to use *low*, *medium* and *high* in the rating, but a more precise rating can be used as in the risk matrix from IEC 61508 in table 2.1. Sometimes it will give an advantage to substitute with numbers when a higher accuracy is needed in the calculation. Scores of one to three is often used, but have some problem. Is the probability of *medium* (2) twice as high as *low* (1), but *high* (3) only 50% as big as *medium* (2)? One solution to this problem is to use 1, 3 and 10, where the intervals are more equal. This is illustrated in table 3.2 with the numbers in the brackets and it is also used in section 3.2.

We will use the same type of rating for the consequence as described for the probability. As we discussed in section 2.1.3, the risk exposure is found by multiplying the consequence and the probability for the risk. The risk matrix in table 3.2 is used to find the risk for the hazards.

<i>Probability</i>	<i>Consequence</i>		
	High (10)	Medium (3)	Low (1)
High (10)	High (100)	High (30)	Medium (10)
Medium (3)	High (30)	Medium (9)	Low (3)
Low (1)	Medium (10)	Low (3)	Low (1)

Table 3.2: A risk matrix for the risk exposure.

When the hazard and the risk exposure for the hazard are found, it is important to consider actions to reduce the risk. If the risk exposure is high, we should try to change the code or implement a barrier [27]. As discussed in section 2.3.4, a barrier will prevent the hazard from occurring. A barrier will, however, also make the system more complex and will cause extra lines of codes with potential faults. Implementing the barrier will need extra resources from the developers, and may take away time from more important task. For each hazard, the cost of implementing a barrier has to be considered against the possible benefits by reducing the risk for the hazard. A barrier should be implemented if the benefits are higher than the cost of implementing it. The complexity of the barrier and how easy it will be to test it, are also important issues to consider. We will not formalize these aspects in our method, but it can be included in further extensions.

How to test that the hazard does not occur must be considered. This can lead to one or many tests for each hazard and are included in the list of tests for the system in section

3.3. If a barrier is proposed, it is important to also describe how it can be tested. How to test barriers is already discussed in section 2.3.4.

<i>Hazard</i>	<i>Cause</i>	<i>Unit</i>	<i>Probability</i>	<i>Consequence</i>	<i>Risk exposure</i>	<i>Action/Barrier</i>	<i>Tests</i>
Can not calculate Y	The variable X has a illegal value	Module 1	Medium	High	High	Prevent that variable X will get a illegal value when used in module 1	Check if X can have an illegal value.

Table 3.3: A table with the information obtained in the hazard analysis.

In table 3.3 we have show the information we need about a hazard. The risk exposure is found from the probability and consequence using the matrix in table 3.2. A barrier and test for the hazard is proposed. The barrier can be found by analyzing the description and cause of the hazard. The developers will have the responsibility to implement the barrier, whereas the testers must test that it is working.

3.2 Prioritizing the modules in the system

In this section we will describe how to estimate the risk for each module in a system. We discussed the risk categories affecting the cost of failure and the defect generators from the process and the code in section 2.4. These factors will be used in our method and can be found in table 3.4. We will use the risk exposure formula described in section 2.1 with cost used instead of consequence. Our method is similar to the methods used by Amland [2] and Schaefer [21].

<i>Cost factors</i>	<i>Defect generators from the process</i>	<i>Defect generators in the code</i>
Criticality	New methods	Complexity
Visibility	New tools or techniques	Number of faults
Most used	Number of people involved	Changes
Reputation	Time pressure	Design quality
Maintainability	Optimization	
Legal consequences		
Number of hazards		

Table 3.4: Risk categoriees. The risk factors used in the risk analysis.

For each project it is important to decide early in the process what risk factors to use. All risk factors (cost factors and defect generators) may not be relevant to the project. Some risk factor may also be too difficult or too costly to assess. We will propose to use the roughly same number of cost factors and defect generators. This will give a balance between consequence and probability. If more defect generators are chosen, the probability of defects will have a greater influence on the risk exposure. Some risk factors may be skipped later when the risk analysis is performed, if it is difficult to collect that accurate data for all the modules. There is also a limit to how many risk categories we will need to perform a useful risk analysis. If too many risk factors are chosen, it may be seen as a waste of time and it is too much effort to consider every category thoroughly. This may affect the quality of the collected data. Thus, only the most important risk categories for that project should be chosen. For each risk factor a weight is given. This weight reflects the importance of the risk factor in the analysis. In the software tool we will use a rating of *low*, *medium* and *high* in the user interface to simplify for the user, but to illustrate how to calculate the risk we will substitute them with the values 1, 3 and 10 in this section.

Each module in the system has to be analyzed. We will start by estimating the cost number. This number indicates the cost or consequence if there are many faults and problems in the module. For each of the chosen cost factors a value will be given for how relevant this risk is for that particular module. We will use the numbers 1, 3 and 10. The cost number is found by using the formula below. This is illustrated inside the brackets under cost number in table 3.5. The **bold** numbers are weights given to each cost factor. Also take a look at table 3.7 for a summary of how the cost number is found.

$$\text{Cost number} = \sum(\text{cost (weight)} * \text{cost (value)})$$

It will give an advantage to do the cost analysis together with customers and users. They will have a better knowledge about the consequences if the system fails.

<i>Cost factors</i>	<i>Maintenance</i>	<i>Visibility</i>	<i>Reputation</i>	<i>Cost number</i>
Weights	3	3	10	
Module 1	1	1	3	36 (3 *1+ 3 *1+ 10 *3)
Module 2	10	10	3	90 (3 *10+ 3 *10+ 10 *3)

Table 3.5: Calculation of the cost number.

The probability number is found in the same way as the cost number. The defect generator from the process and the code are used together to find the probability. The probability number is found using the formula below.

$$\text{Probability number} = \sum(\text{defect generator (weight)} * \text{defect generator (value)})$$

Some of the defect generators, for instance complexity, are useful to collect just after implementation of each module. The information may be selected by the testers or the developer responsible for implementing each module. The developers have a better understanding of each module and can assess them easier. A problem is that some developers may give their modules lower or higher values than other developer would have done. If the testers participate in the process, they may check that the developers use the same measures. Furthermore, if all the developers use the same detailed description on how to valuate the modules, the measurement errors will be reduced.

	<i>Defect Generators</i>			<i>Probability number</i>
	<i>Complexity</i>	<i>Changes</i>	<i>New techniques</i>	
<i>Weights</i>	10	1	3	
Module 1	3	1	3	40 (10*3 + 1*1 + 3*3)
Module 2	10	3	3	112 (10*10 + 1*3 + 3*3)

Table 3.6: Calculation of the probability number.

<p>Step 1: Chose the risk factors to use in the risk analysis Step 2: Give a weight to each of the chosen risk factors Step 3: Select a module Step 4: Give a value for each risk factor for that module Step 5: Multiply the value from step 4 and the weight from step 2 for each risk factor Step 6: Add all products found in step 5 to find the cost/probability number for that module Step 7: Repeat step 3-6 until the cost/probability numbers for all the modules are found</p>
--

Table 3.7: The process used to find the cost number and the probability number

From the cost number and the probability number, we can find the risk exposure number for each module as follows:

$$\text{Risk Exposure} = \text{Cost number} * \text{Probability number}$$

When the risk exposure is found for all the modules, the modules are collected in a list, sorted by risk exposure. This list will show the modules with the highest risk and more

resources need to be spent on these modules. Since the risk exposure number depends on how many risk categories are used and what weights they are give, the risk exposure will vary from project to project. The highest risk exposure number in one project can be the lowest in another. The aim of the risk analysis is to compare the risk of the modules within on project to prioritize the tests. Because of this, we will sort the modules according to the risk exposure number and give the highest one third *High*, the next one third *Medium* and the lowest one third *Low* risk exposure. The number of modules will not always be dividable by three. In table 3.8 we have illustrated this with one *Medium* more than *High* and *Low*. This will give an average of *Medium* and a distribution as equal as possible.

<i>Modules</i>	<i>Cost number</i>	<i>Probability number</i>	<i>Risk exposure number</i>	<i>Risk exposure</i>
Module 2	90	112	10080	High
Module 6	160	42	6720	High
Module 4	42	131	5502	Medium
Module 5	42	61	2561	Medium
Module 7	42	36	1512	Medium
Module 1	36	40	1440	Low
Module 3	23	23	506	Low

Table 3.8: The Risk exposure is calculated.

3.3 Prioritizing the tests

In section 3.1 we proposed how to create tests from hazards. We have also discussed how to design tests in section 2.2. In our method for risk-based testing we focus on system testing. The proposed method will help us to select the right tests to minimize the total system risk. Tests related to the hazards with high risk exposure from section 3.1 and the tests for the most risky modules from section 3.2 will be prioritized. In this section we will show how to do this.

A test can address a specific hazard or test that a barrier is working. These tests will use the risk for the related hazard found in chapter 3.1 as the test risk. For the rest of the tests, the module that it is testing must be identified. If the test is testing more than one module, the most relevant should be chosen. To choose the module with the highest risk exposure will is also a possibility. This will ensure that the test will not get too low priority. The risk exposure of this module is used as the test risk. Since all tests will not have the same importance and ability to identify faults, we introduce a test efficiency value. The testers must give a *low*, *medium* or *high* rating to each test for how confident the tester is that the

test will find faults. The tester should also consider the resources needed for running the test. A test that will not need much time for running should be given a higher efficiency. When the risk and efficiency for a test is valuated, the matrix in table 3.9 is used to find the test priority.

Module/Hazard/Barrier Risk Exposure	Test efficiency		
	High	Medium	Low
High	Very high	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Very low

Table 3.9: Risk matrix used in the final test priority.

Table 3.10 illustrates how a set of tests are prioritized based on risk and efficiency. “Test 4” is testing a barrier for “Hazard 1”. It has the highest priority since “Hazard 1” has a high risk and the test has *High* efficiency. “Test 1” is using the risk of “Module 4”, which is also *High*, but the efficiency is only *Medium*. Because of this it will not be given as high a priority as “Test 4”. “Test 2” and “Test 3” both test “Hazard 2”. The difference in priority is due to the fact that the “Test 2” is not very efficient when testing that the barrier for the hazard is working.

Test name	Type of test	Module/ Hazard	Risk	Test efficiency	Test priority
Test 4	Barrier	Hazard 1	High	High	Very High
Test 1	Module	Module 4	Medium	High	High
Test 3	Hazard	Hazard 2	Medium	Medium	Medium
Test 2	Barrier	Hazard 2	Medium	Low	Low

Table 3.10: A prioritized list of tests.

The tester should at least perform all the tests with *Very high* and *High* priority and also most of the tests with *Medium* priority. Hutcheson [9] states that execution of 67 % of the most important tests will normally be enough to release the product with an acceptable risk. The resources for each test must also be estimated. That is an estimate of how much time in minutes we will need to run each test. This should include the set up time and the time needed to clean up the databases after each test. When deciding when we can release the system or if more testing is needed we have to look at the cost of not to test versus the benefits of more testing [27]. When the testers can agree with the management that the

benefits of releasing the product are higher than the risk of not testing any more, we are ready to release the product.

4. Requirement specification for the software tool

The method described in the previous chapter can be performed by using a spreadsheet, but this will probably become complicated. To implement a software tool to support the method will help the testers, without worrying about calculations. It will be easier to collect and store information about modules, hazards, barriers and tests. A software tool can give all the testers easy access to the data. The tool described in the requirements is not intended to support every aspect of testing. It will help the tester to gather relevant information about the risk analysis so that the tests can be prioritized. User access and keeping track of the testing process is not prioritized in this version, but the tool can easily be extended with this functionality.

4.1 Functional requirements

The functional requirements describe what the user should be able to do with the software tool.

4.1.1 Project

Since most organizations have more than one project running at the same time, the software tool must be able to handle many projects.

- F-1: The user must be able to create a new project.
- F-2: The user must be able to give the project a name.
- F-3: The user must be able to select a project.

4.1.2 Risk category

The risk category describes factors that will influence the risk related to the modules in the system. The cost factors and defect generators are the two types of risk categories used in the software tool. The system will contain a predefined set of these. When a new project is created, the risk categories to be used in that particular project have to be selected and given weights.

- F-4: The user must be able to create a new risk category.
- F-5: The risk category must be given a type, cost factor or defect generator.
- F-6: The user must be able to modify an existing category.
- F-7: The user must be able to delete a risk category.
- F-8: The user must be able to give each risk category a name, description and type.
- F-9: The user must be able to select to risk categories predefined in the software tool to be used in that particular project.
- F-10: The user must be able to give each risk category a weight: high, medium or low.

4.1.3 Module

Each module is given a ranking for each risk category. The ranking and weight are internally substituted by a number to perform the risk number calculation. This is described in section 3.2. The modules are sorted by the risk exposure number and as equal number of high, medium and low ranking as possible are given to the modules. The user can create a set of tests for each module.

- F-11: The user must be able to create a new system module.
- F-12: The user must be able to modify an existing system module.
- F-13: The user must be able to delete a system module.
- F-15: The user must be able to give each system module a ranking for each of the selected risk categories: high, medium or low.
- F-14: The user must be able to give each module a name and a description.
- F-16: The user must be able to see a list of all the modules sorted by risk, high, medium or low.

4.1.4 Hazard

For each hazard a risk exposure is calculated using the risk matrix shown in table 3.2. The user can create a set of barriers and tests for each hazard.

- F-17: The user must be able to create a new hazard.
- F-18: The user must be able to modify an existing hazard.
- F-19: The user must be able to delete a hazard.
- F-20: The user must be able to give each hazard a name, description and cause.
- F-21: The user must be able to give each hazard a probability and a consequence: high, medium or low.
- F-22: The user must be able to get a list of all the hazards sorted by risk, high, medium or low.

4.1.5 Test

A test can be created to test a module, a hazard or a barrier. Each test can only belong to one of the modules, hazards or barriers, but each module, hazard or barrier can have many tests. The tests are sorted and prioritized using the risk matrix shown in table 3.9.

- F-23: The user must be able to create a new test.
- F-24: The user must be able to modify an existing test.
- F-25: The user must be able to delete a test.
- F-26: The user must be able to give each test a name, description and a related module, hazard or barrier.
- F-27: The user must be able to give a test an efficiency: high, medium or low.

F-28: The user must be able to get a list of all the tests sorted by priority, very high, high, medium, low or very low.

4.1.6 Barrier

The barrier will prevent a hazard. It must be related to one specific hazard. The user can create a set of tests for each barrier.

F-29: The user must be able to create a new barrier.

F-31: The user must be able to delete a barrier.

F-30: The user must be able to modify an existing barrier.

F-32: The user must be able to give each test a name, description.

F-33: The user must be able to see a list of all the barriers sorted by risk, high, medium or low.

4.2 Non-functional requirements

Non-functional requirements describe quality, the environment and technology to be used, the project plan and the development methods. We will discuss a few quality characteristics and the technology used. Since the project is small, the project plan and development methods are not described.

4.2.1 Quality

ISO 9126 [12] defines a set of software quality characteristics - functionality, reliability, usability, efficiency, maintainability and portability. Usability is the most important of these characteristics for this software tool. It must be easy to use and to learn. The system should be stable and should not lose any data.

4.2.1 Technology

The software tool will be implemented using ASP.NET. This will give the users the possibility to access the application using intranet or Internet.

5. Presentation of the software tool

We have implemented the software tool from the requirements presented in chapter 4. First the implementation is discussed, before we describe how it works.

5.1 Implementation

The software tool was implemented as web pages using ASP.NET and C# as programming language. In section 5.2 we will describe each of these web pages and what functional requirement they cover. All of these requirements are covered by the implemented software tool.

As mentioned in 4.2 the most important non-functional requirements are usability and that it should be easy to learn. This has been emphasized during implementation by using datagrids in most of the pages. A datagrid presents data in a table and gives the user the possibility to easily work with the data doing things like editing, deleting or adding rows. This has also kept the number of pages low, thus making it easier to understand how the software tool works. Descriptions on how to use the tool has also been added to many of the pages to help the user the first times the software tool is used.

5.1.1 Data model

The data model presented in figure 5.1 was implemented in the database for the software tool. The subtypes of test and risk category were not implemented as tables. Defect generator and cost factor use the risk category table with type as an extra field. The same was done for the three different kinds of tests where id and type indicate the foreign key. The web pages access the database tables each time the pages are generated.

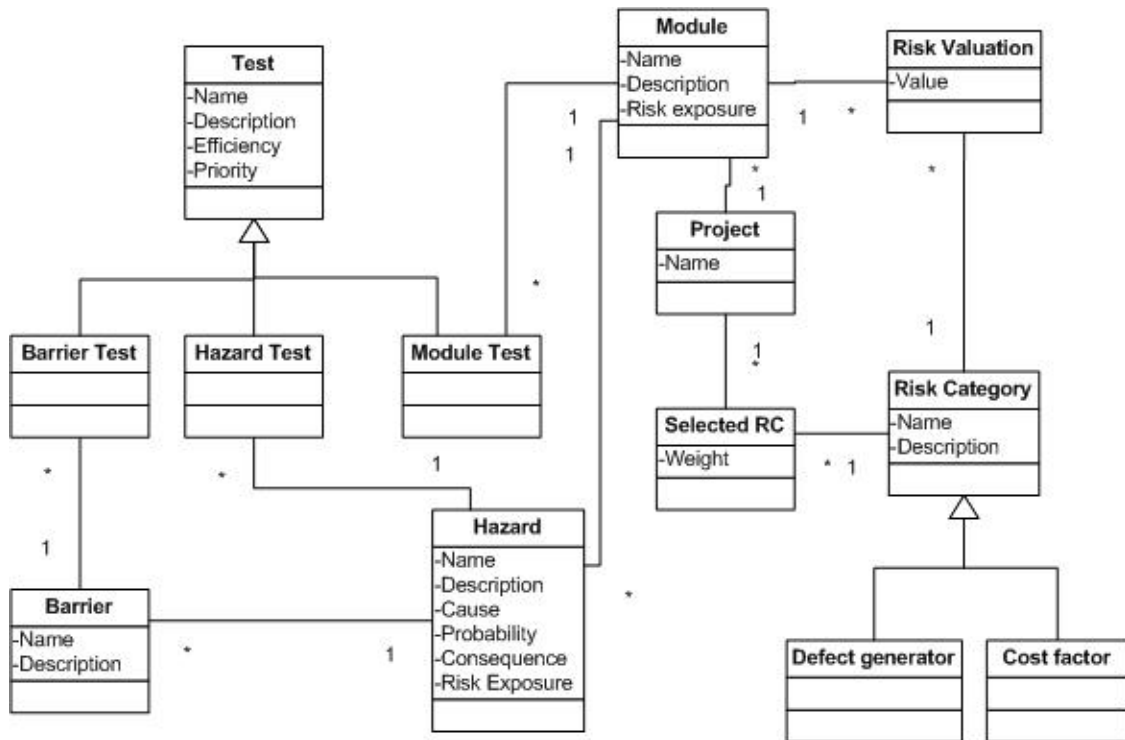


Figure 5.1: A UML class diagram showing the data model for the software tool.

5.2 Screen shots

The software tool consists of nine web pages. We will describe and present the screen shots for each of them in this section. The software tool is only creating references to projects, modules, tests, hazards and barriers to be able to prioritize the tests. In the rest of this chapter we will always mean “create a reference” when the term “create” is used. To be able to use the software tool it is important to be familiar to the terms presented in this report, as “hazard” and “barrier”. It is also an advantage if the user understands how the method described in section 3 is working. Below is a list what the tester must have prepared before the software tool is used.

1. The user has to consider what risk categories (table 3.4) that are relevant to the project and to give each of them an importance weight.
2. The user must have divided the system up into modules to test.
3. The user must define tests related to the modules.
4. A hazard analysis (HazOp, FMEA or PHE) must have been carried out.
5. For each hazard identified, the user has to consider if a barrier should be implemented.
6. For the hazards where barriers will be implemented, tests verifying that the barriers are working must be defined.
7. For the hazards with no barriers, tests checking if the hazards will occur must be defined.

5.2.1 Select project

The Start.aspx page is the first page the user will meet when he wants to use the test tool. In this page the user can select a project from a list of all projects in the database. When a project is selected a new session is started and the selected project is used in the following pages. It is also possible to create a new project, by pressing the “New project” button. This will open the page NewProject.aspx as presented in the next section. The project name cannot be changed and the project can not be deleted. This functionally can, however, easily be implemented, but was not prioritized. The page covers the functional requirement: F-3.

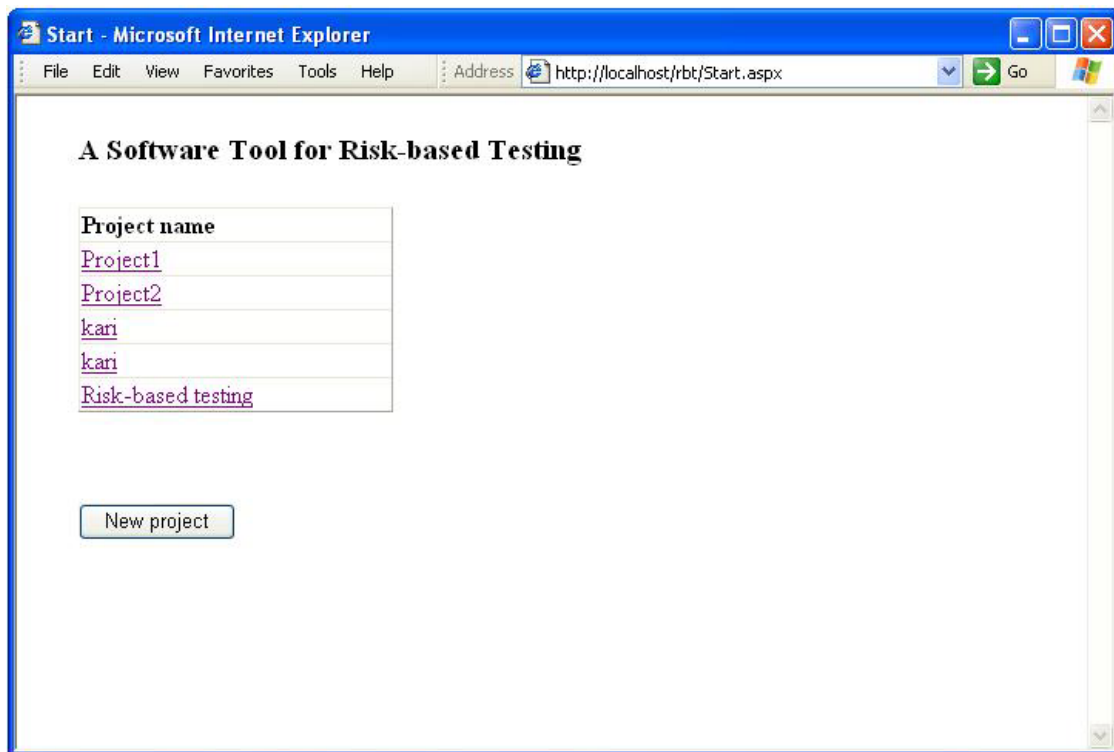


Figure 5.2: Screen shot of Start.aspx.

5.2.2 Create a new Project

NewProject.aspx gives the user the possibility to create a new project. A new project is created by clicking the “OK” button. A new session is started with this project. The next page is RiskCategory.aspx described in next section. The user can chose to “Cancel” and will then be brought back to the start page. The page covers the functional requirements: F-1 and F-2.

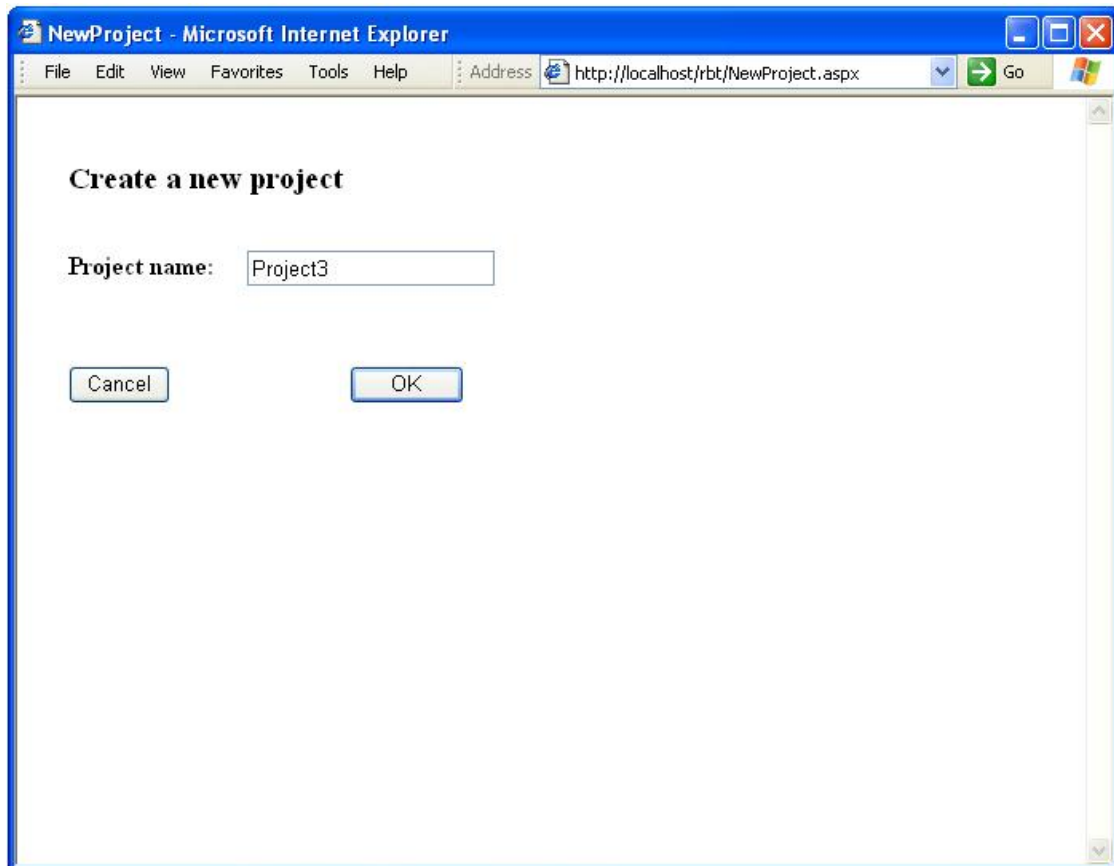


Figure 5.3: Screen shot of NewProject.aspx.

5.2.3 Select and weight risk categories

In RiskCategory.aspx all cost factors and defect generators are presented in two datagrids. The name, description and weights for each risk category are listed. The user can give a *Low*, *Medium* or *High* weight to each risk category depending of how important it is for the selected project in Start.aspx. If *Not selected* is chosen, the risk category will not be used in this project. It is also possible to edit the name and description for each risk factor and delete one or more risk categories from the project. New risk categories can be added by filling out text fields at the bottom of each datagrid and press the “Add” link. The page covers the functional requirements: F-4, F-5, F-6, F-7, F-8, F-9 and F-10.

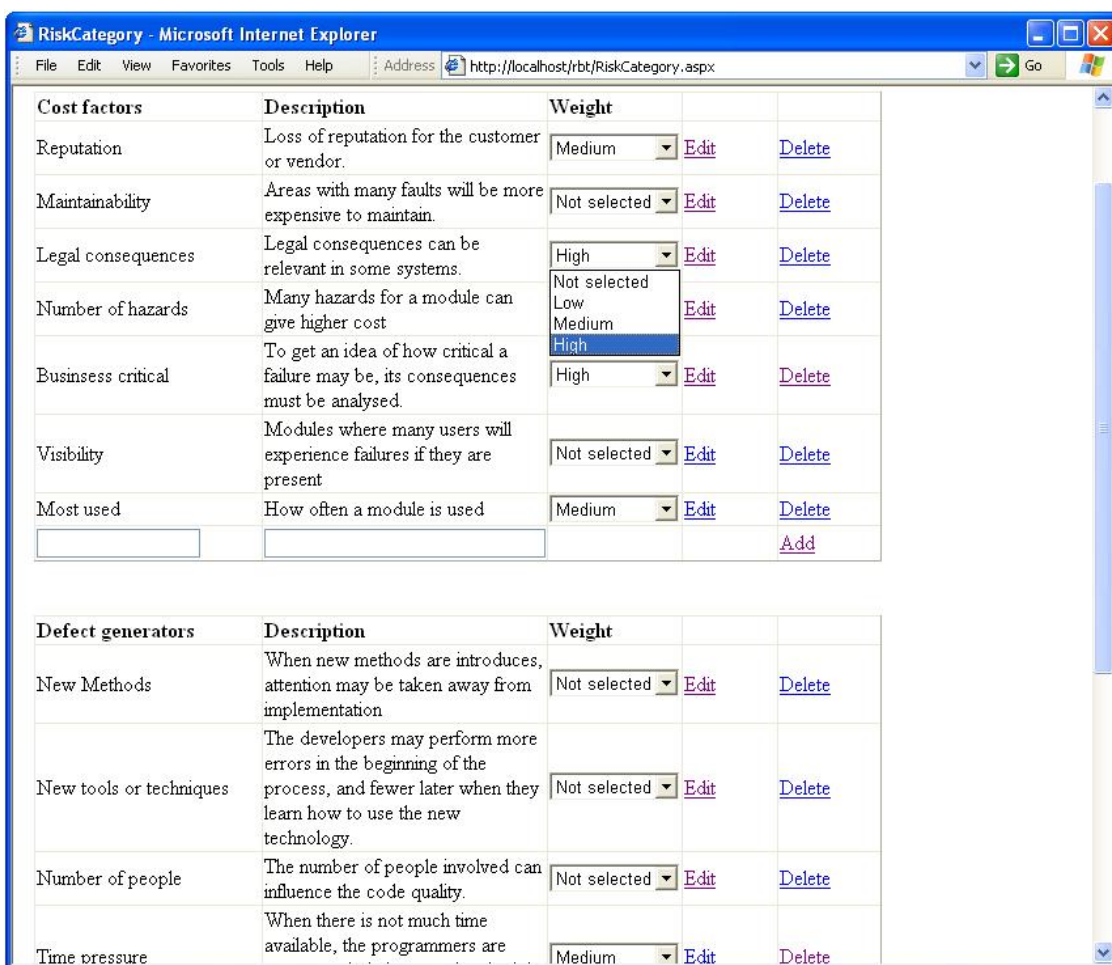


Figure 5.4: Screen shot of RiskCategory.aspx.

5.2.4 List the modules

Modules.aspx lists all the modules the user creates for the selected project in a datagrid. The user can edit the name and description for each module or delete one or more modules from the project. New modules can be added using the text fields at the bottom of the datagrid. To find the risk exposure for a module, the module has to be valuated in ValuatedModules.aspx presented in the next section. The “Give values” link in the row for that module will bring the user to ValuateModules.aspx. If no values are given, the cell will contain “NA” - not available. The risk exposure is calculated as described in section 3.2. It is also possible to add a test for the module using the “Add test” link, linking to CreateTest.aspx. The “Add hazard” link brings the user to Hazards.aspx where hazards for that module can be created. The “OK” button brings the user back to Main.aspx. Modules.aspx covers the functional requirements: F-11, F-12, F-13, F-14 and F-16.

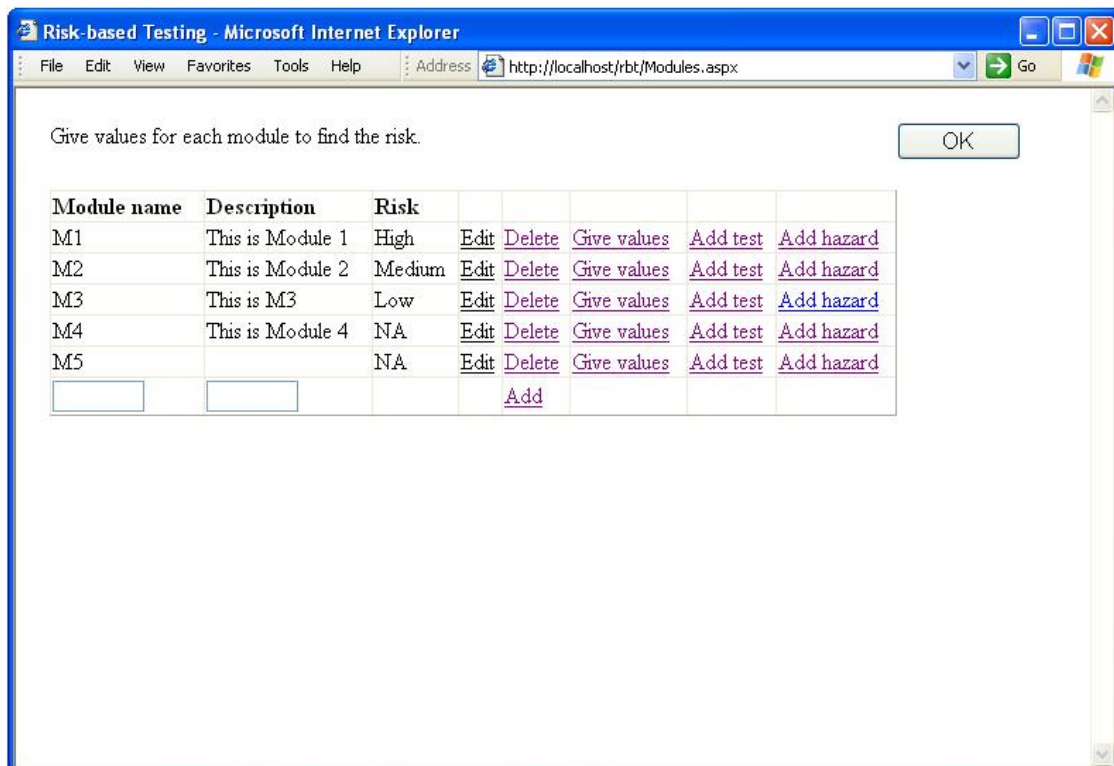
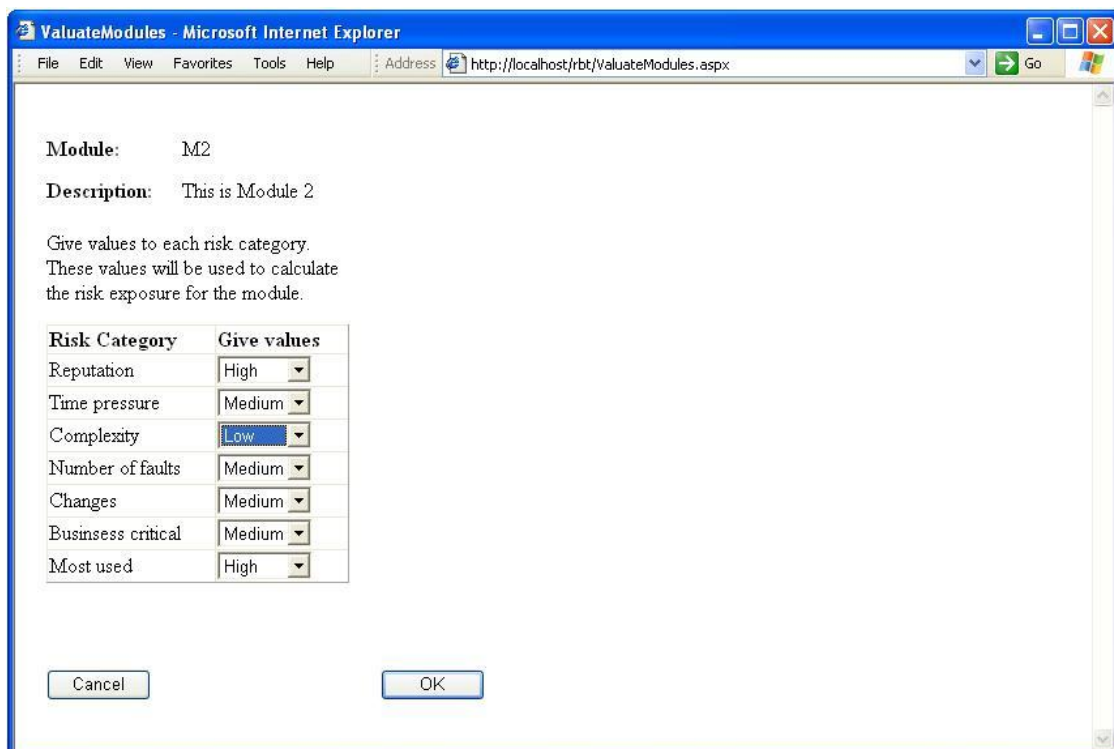


Figure 5.5: Screen shot of Modules.aspx.

5.2.5 Valuation of the modules

In ValuateModules.aspx a module is analysed. The name and description for the module to be analysed is presented at the top. In the datagrid all the risk categories selected in RiskCategories.aspx are listed. Each of these must be given *Low*, *Medium* or *High* ranking depending on how relevant that risk is for the module. The “Cancel” and “OK” buttons bring the user back to the module page. The “OK” button saves the values in the database and calculates the risk exposure for that module. The page covers the functional requirement: F-15.



Module: M2

Description: This is Module 2

Give values to each risk category.
These values will be used to calculate the risk exposure for the module.

Risk Category	Give values
Reputation	High
Time pressure	Medium
Complexity	Low
Number of faults	Medium
Changes	Medium
Business critical	Medium
Most used	High

Cancel OK

Figure 5.6: Screen shot of ValuateModules.aspx.

5.2.6 Create a new test

CreateTest.aspx gives the user the opportunity to create a new test. The user must give the test a name and test efficiency. A related module, hazard or barrier must also be chosen. If the “Add test” links in one of the module, hazard or barrier pages are chosen, this is already chosen from the list. The user can also give a description of the test. This page is also used when a test is edited from the test datagrid in the main page. The page covers the functional requirements: F-23, F-24, F-25, F-26 and F-27.

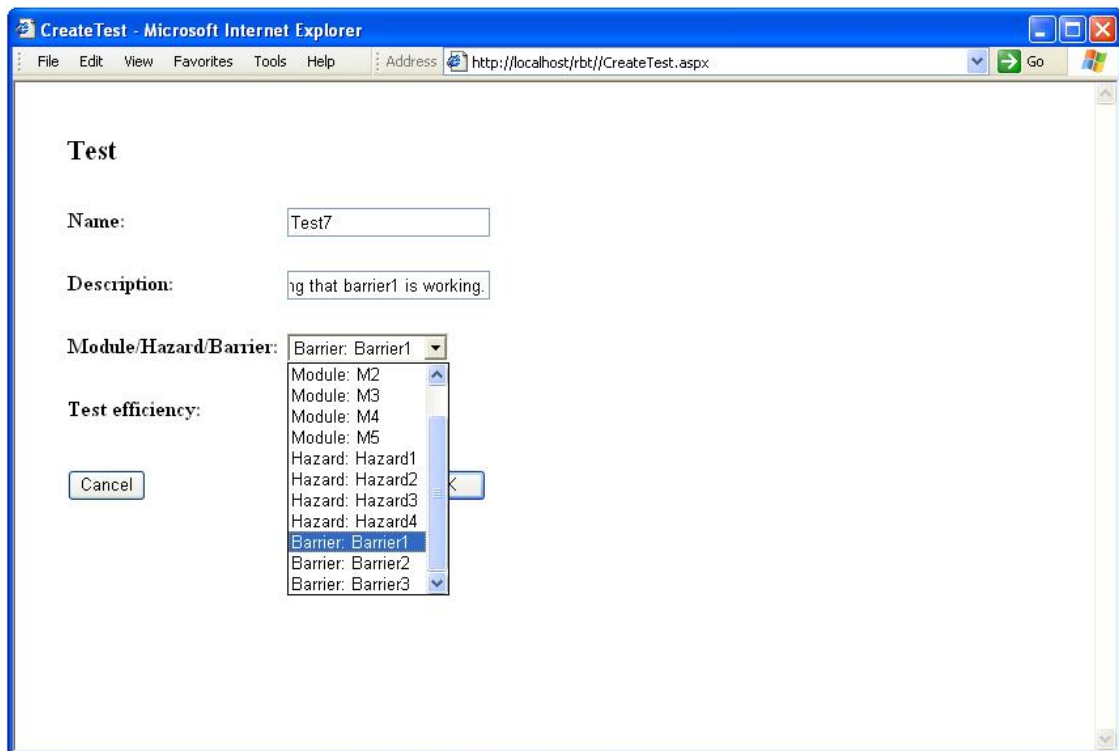
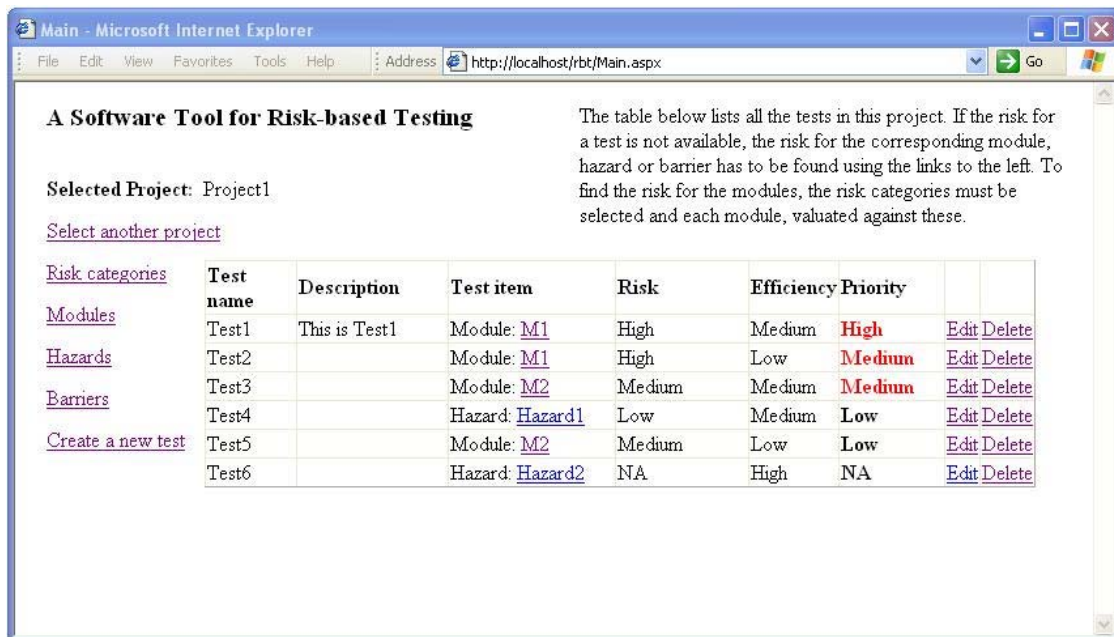


Figure 5.8: Screen shot of CreateTest.aspx.

5.2.7 List the tests

This is the main page with links to the other pages. The priority for the test presented on this page is the most important output from the software tool. The name, description and related the module, hazard or barrier for each test is also listed. The priority is calculated from the related module, hazard or barrier risk and the test efficiency for that test using the matrix in table 3.9. The user can also edit or delete a test, pressing the links on that row. The page covers the functional requirement: F-28.



A Software Tool for Risk-based Testing

The table below lists all the tests in this project. If the risk for a test is not available, the risk for the corresponding module, hazard or barrier has to be found using the links to the left. To find the risk for the modules, the risk categories must be selected and each module, valuated against these.

Selected Project: Project1

[Select another project](#)

[Risk categories](#)

[Modules](#)

[Hazards](#)

[Barriers](#)

[Create a new test](#)

Test name	Description	Test item	Risk	Efficiency	Priority		
Test1	This is Test 1	Module: M1	High	Medium	High	Edit	Delete
Test2		Module: M1	High	Low	Medium	Edit	Delete
Test3		Module: M2	Medium	Medium	Medium	Edit	Delete
Test4		Hazard: Hazard1	Low	Medium	Low	Edit	Delete
Test5		Module: M2	Medium	Low	Low	Edit	Delete
Test6		Hazard: Hazard2	NA	High	NA	Edit	Delete

Figure 5.7: Screen shot of Main.aspx.

5.2.8 List the hazards

Hazards.aspx implements the method described in section 3.1. The page lists all the hazards. Each hazard in the datagrid consists of a name, description, cause, related module, probability and consequence. The Risk is found from the probability and consequence using the matrix in figure 3.2. The hazards can be edited and deleted using the “Edit” and “Delete” link. New hazards can be added at the bottom of the datagrid. It is also possible to add a test for the hazard using the “Add test” link, linking to CreateTest.aspx. The “Add barrier” link links to Barrier.aspx described in the next section. The page covers the functional requirements: F-17, F-18, F-19, F-20, F-21 and F-22.

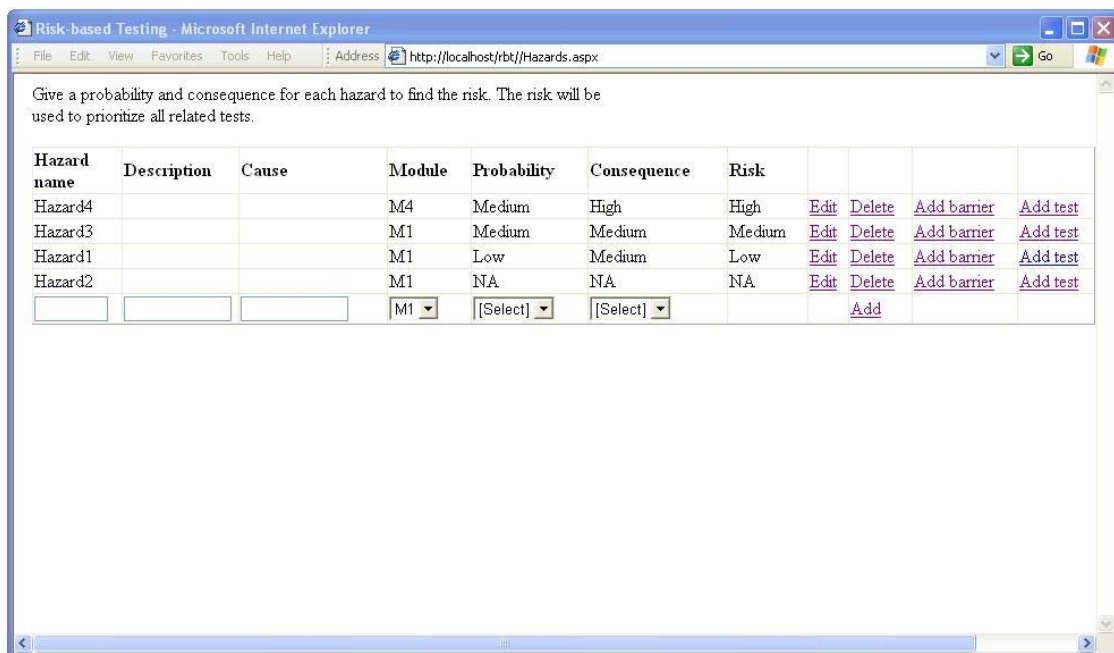


Figure 5.9: Screen shot of Hazards.aspx.

5.2.9 List the barriers

Barrier.aspx gives the user the possibility to create barriers for each hazard. The barrier is an action to prevent the hazard from happening. A name and description are given to the barrier. The user can edit and delete a barrier by using the links on the row the user wants to edit or delete. New barriers can be added at the bottom of the datagrid. It is also possible to add a test, that is testing that the barrier is working by using the “Add test” link. This is linked to CreateTest.aspx. The “OK” button brings the user to Main.aspx. Barrier.aspx covers the functional requirements: F-29, F-30, F-31, F-32 and F-33.

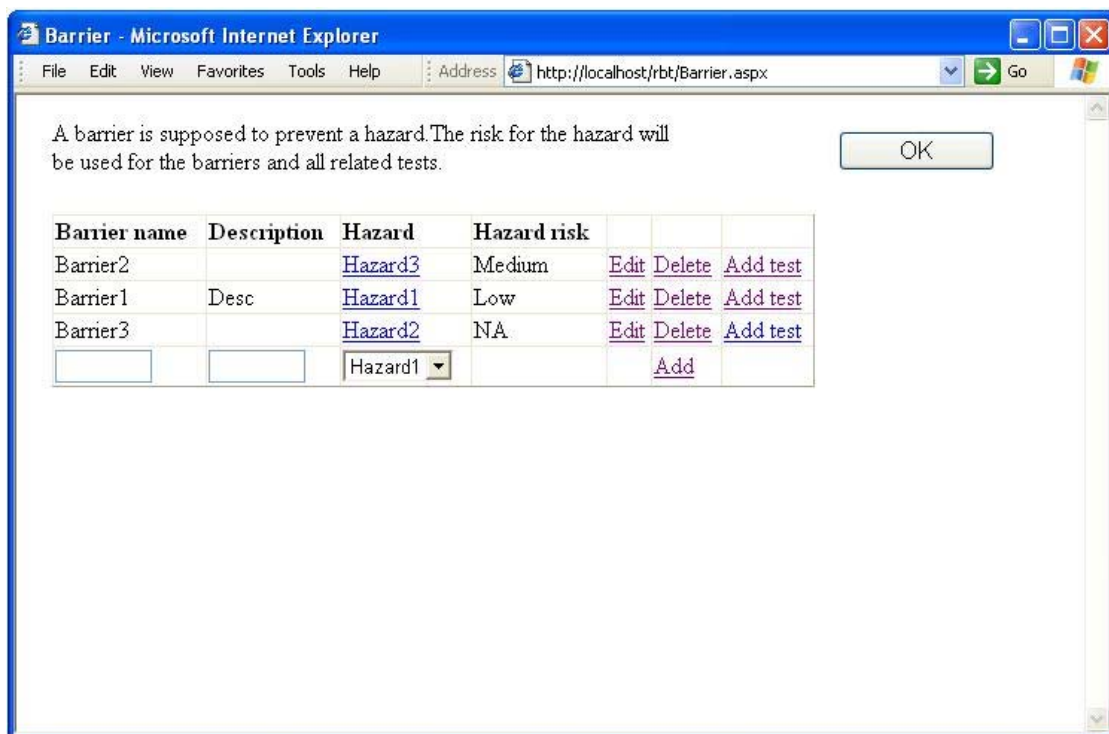


Figure 5.10: Screen shot of Barrier.aspx.

5.3 Trying out the software tool

The tool has been tried out by an experienced tester of large systems and with experience of developing tools for testing. This person had some problems when trying to use the software tool. The main problem seemed to be that the tester was not familiar with the terms used by the software tool and how the software tool was supposed to be used. More explanations and guidance are needed. There were problems to understand the relations between the things done and why they were done. This can be solved by writing a manual, include comments in the software tool or including a help file. A presentation for new users on how to use the software tool and the concepts related to it also seem to be necessary. We have added a few more explanations in the software tool, but more is still needed.

There were also some problems related to the user interface. Some of these have been modified in the presentation in section 5.2. The tester demanded the possibility to modify the risk matrix determining the module risk. Moreover, it should be easy to say that something applying for module X should also apply for module Y and Z. This functionality has not been added due to limitation of time, but can be implemented in further versions of the software tool.

The feedback from the tester was not surprising since the software tool is only a prototype to prove the concepts. The aim was not to implement a commercial software tool.

6. Conclusion

As pointed out in the “State of the Art” chapter, the term risk-based testing has been used in several ways. Sometimes the term has been used for testing where risks has been used to find the tests, but not in a formal way. Other methods prioritize the tests by using the risk for each part of the system it is testing or the hazard the test is addressing. In this report we have presented a method for risk-based testing and tried to use the best of some of several methods. The prioritizing of subsystems described by Amland [2] and Schaefer [21] has been used in our method. Prioritizing tests related to hazards described by Gerrard [6] has been extended by also applying tests for barriers. These two approaches have been combined to prioritize the tests in our method.

Furthermore, we have shown how this can be supported by a software tool. The use of a software tool will help the tester to perform and systematise the information needed for the method. A web application will give easy access to all information gathered by the tool. The software tool has been tried out by an experienced tester. The software tool still needs some modifications and improvements before it can be used in the industry. Thus, the implementation can be considered as a proof of concept.

7. Further work

Further work includes improving the method described in chapter 3, extending the functionality of the software tool and trying out the software tool.

7.1 Improvements of the method

The tester that tried out the software tool pointed out that even as an experienced tester it is difficult to determine the efficiency for a test before it has been run. The test efficiency is a subjective evaluation of the test's ability to identify faults in a module. For a test related to a barrier it is the ability to test that the barrier is working and prevents the related hazard. A test related to a hazard tests if the hazard can occur. A way better to estimate the test efficiency is necessary for prioritizing the tests. The test efficiency should also consider the resources a test needs for running. Some tests may not need much time to be executed, but may have a low priority. Still these tests will probably be worth running. To include the resources needed for each test, and use this when the tests are prioritized will at least partly solve this problem.

The relations between a module and its hazards were not used for any purpose. If a hazard is related to a module with high risk, it will probably cause more problems than if it was related to a module with a low risk. The risk of the module can be included when hazard's risk is calculated.

To prioritize the tests verify that a barrier is working, the risk of the hazard it is preventing is used. Since barriers are implemented as code, they will have the same risk factors as the modules. Because of this the barriers should be valued in the same way as the modules (see section 3.2) to find their risk exposure.

7.2 Extending the functionality of the software tool

The software tool is only a "proof of concept" and will require some extensions to be used in the industry. In section 5.3 we have already discussed some of the improvement the software tool will need. We discuss further improvements in this section.

Implementing user accounts and several access levels can be useful. This will give the opportunity to note who is responsible for testing each module and is responsible for collect information needed to perform the analysis. Keeping track of the status of the tests and resources needed for testing will help the testers easier to plan and perform the testing process.

A problem during risk analysis is that it is difficult to imagine think that may go wrong. The methods described do not help to identify hazards that are not yet in one of the participants' head [25]. This is a challenge. Hazards found in earlier project may happen

in the new project as well. Also the barriers and tests from earlier project can be reused in this fashion.

7.3 Try out the software tool

There has not been enough time to try the software tool in a real project and it is thus difficult to know how useful such a tool will be. A case study should be performed in an organization already focusing on risk and hazard analysis. They will easier be able to decide whether the software tool is useful or not.

References

- [1] Ed Adams, Optimising preventive service of software products, IBM J. Research and Development, 28, 1, page 2-14, 1984
- [2] Ståle Amland, Risk Based Testing and Metrics, EuroSTAR '99, Barcelona, November 1999
- [3] Tom Anderson, Mei Feng, Steve Riddle, Alexander Romanovsky, Protective Wrapper Development: A Case Study, 2nd International Conference on COTS-Based Software Systems, Ottawa, Canada, February 2003
- [4] James Bach, Heuristic Risk-Based Testing, Software Testing and Quality Magazine, 11/99, November 1999
- [5] Yanping Chen, Robert L. Probert, A Risk-based Regression Test Selection Strategy, Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research, Toronto, Ontario, Canada, 2003
- [6] Paul Gerrard, Risk-Based E-Business Testing, 2002, ISBN 1580533140, page 3 – 29 and 51 - 80
- [7] Anup K. Ghosh and Jeffrey M. Voas, Inoculating software for survivability, Communication of the ACM Vol. 42. No.7, July 1999
- [8] Jim Heumann, Generating Test Cases from Use Cases, The Rational Edge 06/01, 2001
- [9] Marnie L. Hutcheson, Software Testing Fundamentals, 2003, ISBN 0-471-43020-X
- [10] IEEE Std 829-1998
- [11] IEC 61508-5, Functional safety of electrical/electronic/programmable electronic safety-related systems, 1998
- [12] Draft for CD, ISO/IEC 9126-1, September 2004
- [13] Stephen H. Kan, Metrics and Models in Software Engineering, second edition, 2003, ISBN 0-201-72915-6, page 119
- [14] Cem Kaner, What is a Good Test Case, STAR East, May 2003
- [15] Cem Kaner, Jack Falk, Hung Quoc Nguyen, Testing Computer Software, 1999, ISBN 0-471-35846-0
- [16] Philippe Kruchten, The Rational Unified Process an Introduction, third edition, 2003, ISBN 0-321-19770-4, page 119
- [17] Shari Lawrence Pfleeger, Risky Business: What we have yet to learn about Risk Management, Journal of Systems and Software, Vol. 53, 2000 page 265-273
- [18] Timothy Letherbridge, Object-Oriented Software Engineering, 2002, ISBN 0-07-122689-3, page 110 – 124, 234, 373 – 375
- [19] Glenford J. Myers, The art of software testing, 1979, ISBN: 0-471-04328-1

- [20] Marvin Rausand, Risiko Analyse veiledning til NS 5814, 1991, ISBN 82-519-0970-8, page 41 - 100
- [21] Hans Schaefer, Strategies for Prioritizing Tests, STAR WEST, 1998
- [22] Shamus P. Smith, Michael D. Harrison and Bastiaan A Schupp, How explicit are the barriers to failure in safety arguments?, In Proceedings of the SafeComp, September 2004
- [23] Proceedings of the STANZ conference 2002, Wellington 25.-26. November 2002
- [24] Tor Stålhane, Jacques Hérard, Andreas Söderberg, Timo Malm, Kaarle Kylmä, Ilpo Pöyhönen, NT Techn report 460, Approved 2000-04
- [25] Tor Stålhane, The Role of HazOp in Tesing Health-Care Applications, CVSHC, 5th Conference on Software Validation for Health Care, Dusseldorf, Germany, April 2005.
- [26] Tor Stålhane, Risk Based Testing 1 (slides), EVU-Course, Oslo, 2004
- [27] Tor Stålhane, Risk Based Testing 2 (slides), EVU-Course, Oslo, 2004
- [28] Tor Stålhane, Gunhild Sivertsen Sørvig, Risk Analysis as a Prioritizing Mechanism in SPI, EuroSPI 2003
- [29] Tor Stålhane and Kari J. Wedde, The Quest for Reliability – A Case Study, Proceedings 2nd International Conference on Achieving quality in software, Venice, Italy, October 1993
- [30] Roy Patrick Tan and Stephen H. Edwards, An assertion Checking Wrapper Design for Java, SAVBCS '03 Helsinki, Finland, September 2003
- [31] Hans Van Vliet, Software Engineering Principles and Practice, 2000, ISBN 0-471-97508-7, page 14 - 17
- [32] Jeffrey Voas, Containing COTS via Intelligent Software Wrappers, GSAW 99: Ground System Architectures Workshop, California, March 1999
- [33] James A. Whittaker, How to breake Software, 2003, ISBN 0-201-79619-8, page 106
- [34] Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Main_Page , June 2005
- [35] Bill J. Wood, Software Risk Management for Medical Devices, Medical Device & Diagnostic Magazine, January 1999

Glossary

Black box testing: Testing technique where inputs are given and the outputs are observed. The technique focuses on the requirement and is also called functional testing.

COTS: Commercial off-the-shelf is used for software manufactured not for one specific application.

DataGrid: A web control in the ASP.NET framework presenting data in a table.

Defect: See fault

Error: The action of writing incorrect code

Failure: “Lack of ability of a component, equipment, sub system, or system to perform its intended function as designed. Failure may be the result of one or many faults” [34].

Fault: “An abnormal condition or defect at the component, equipment, or sub-system level which may lead to a failure” [34].

FMEA: Failure Modes and Effect Analysis, see section 2.3.2.

Functional requirements: “Describe system features or things the system must do” [34]

Hazard: A term used in evaluating safety - a potential unwanted event.

Haz-Op: Hazard and Operability Analysis see section 2.3.3.

Non-functional requirements: “Describe properties the system must have (e.g. performance, availability, accessibility)” [34].

Proof of concept: “A proof of concept is a short and/or incomplete realization of a certain method or idea(s) to demonstrate its feasibility” [34].

Requirements: Describe what the system should do, but they do not say how those requirements should be implemented.

Risk: A possible future, unwanted event that has negative consequences.

Risk Analysis: “A technique used to identify and assess factors that may jeopardize the success of a project or achieving a goal. This technique also helps define preventive measures to reduce the probability of these factors from occurring and identify countermeasures to successfully deal with these constraints when they develop.” [34]

Risk Exposure: Cost of event multiplied by the probability that event

Test: A set of one or more test cases.

Test Case set of instruction designed to detect a particular class of defect, by bringing about a failure [18].

Testing: A process used to identify the correctness, completeness and quality of developed computer software [34].

White box testing: Checks “that the outputs of a program, given certain inputs, conform to the internal design and implement