Test Case Selection and Prioritization: Risk-Based or Design-Based?
Jussi Kasurinen, Ossi Taipale and Kari Smolander
Lappeenranta University of Technology
FI-53851 Lappeenranta, Finland
+358 400 213 864
jussi.kasurinen | ossi.taipale | kari.smolander @lut.fi

ABSTRACT
The objective of this qualitative study was to observe and empirically study how software organizations decide on which test cases to select for their software projects. As the software test processes are limited in resources such as time or money, a selection process usually exists for tested features. In this study we conducted a survey on 31 software-producing organizations, and interviewed 36 software professionals from 12 focus organizations to gain a better insight into testing practices. Our findings indicated that the basic approaches to test case selection are usually oriented towards two possible objectives. One is the risk-based selection, where the aim is to focus testing on those parts that are too expensive to fix after launch. The other is design-based selection, where the focus is on ensuring that the software is capable of completing the core operations it was designed to do. These results can then be used to develop testing organizations and to identify better practices for test case selection.

Categories and Subject Descriptors
D.2.9 [Software Engineering]: Management - Software quality assurance (SQA) D.2.8 [Software Engineering]: Metrics - Process metrics

General Terms
Management, Design, Human Factors.

Keywords
Software testing, Test case selection, Empirical study, Grounded theory.

1. INTRODUCTION
In the software industry, launching a new product in time makes a big difference in expected revenue [9]. However, before its launch, the software has to be tested, which itself is a costly process that can yield over half of total development costs [18]. In addition, regardless of the investment in testing, it cannot cover everything as the size and complexity for achieving full-coverage testing increase almost exponentially when the size of the tested software product increases [36]. Therefore, in most software projects, the matter of selecting which test cases should be included in the test plan also exist[26]. In reality, the number of cases that can be used depends on testing resources like personnel and time schedule [32], while trying to maximize the testing output to enhance product quality.

Testing practices seem to suffer from several hindrances, like using shortcuts, reducing test time, poor planning and poor testability [7]. The attitude towards testing, culminating in the “Let go – deliver now and correct later” mentality, causes additional expenses that could be avoided with some reasonable investments [33]. In literature, test case selection is considered an important aspect of the test process, actually being one of the central aspects in building and defining test strategy [13, 34]. As limited test resources are usual in practice [32], there has to be some method of deciding what test cases are selected.

In this paper, we studied different approaches on how real-world software producing organizations select their approach to test case selection. Our approach was to apply the grounded theory research method [8, 31], observe the practices of different polar types of companies, identify how companies select their test cases, and explain why they apply this type of approach.

This study continues our studies on software testing practice. Our prior studies have covered such topics as process problems and enhancement strategies [17], testing resources and test automation [15] and outsourcing [16].

The paper is organized as follows: In Section 2 we introduce related research concepts and in Section 3 the approach that was applied in this study. In Section 4 we present our results and their implications are discussed in Section 5. Finally, the paper is closed with conclusions in Section 6.

2. RELATED RESEARCH
The selection of test cases based on costs or related risk is not a novel concept. For example, Huang and Boehm [9] discuss cost evaluation methods for testing. By ranking the test cases based on their value, i.e. the amount of money lost if the test fails, a 20% investment in testing is sufficient to achieve 80% of the software value. Similar results of testing cost-effectiveness have also been reported by Yoo and Harman [38]. Petschenik [25] even argues that testing can be organized effectively even with as low as 15%
of the perceived resource needs, if the resources are focused on critical aspects.

Redmill [26] discusses this concept even further. As complete testing is not possible, it directs the testing process towards selective testing. As for the test case selection approach, there are different methods, which vary in applicability or results, but in general the testers seem to agree on applying risk-based selection [1, 26]. However, the criterion on which the selection is based on is usually incomplete or undefined. This often leads to a solution where risk analysis is based on individual experience and can be biased. For example, for developers the priorities for technical risks may be well-adjusted. However, risks associated to other stakeholders, concepts like legal costs and compensations, loss of reputation for the company or maintainability by third party associates, are probably beyond the scope of a single software developer [26]. A study by Do and Rothermel [4] suggests that ultimately the selection and testing cut-off point is a tradeoff between the costs of applying additional testing versus the costs of missing errors. Therefore it is plausible in real life to cut testing short to keep the deadline, as the loss caused by product delay supersedes the losses caused by releasing error-prone software [9]. Only in such extreme cases as an incomplete implementation of core features or crippling quality issues delaying the deadline can be considered a feasible option [27].

One proposal to help the test process is to provide better testability for product components [17, 20, 21, 35]. The rationale for this action would be that supporting testability would speed up the process of creating test plans and allow easier test case generation. By having clearer objectives for testing and an easier way to ensure test coverage, the effectiveness of testing work could be increased without severe expenses [38]. However, this approach is not as straightforward and easily implementable an improvement as it would seem. In these types of projects, the test strategy becomes crucial, as the enablers for testability have to be implemented into the source code simultaneously with the actual development process. In other words, the developers have to plan the development ahead to make sure that every needed case can be tested [21]. Within software projects, this would require rigid plan-driven development or continuous testability analysis for verification purposes, which would obviously generate other expenses [21, 35]. In contrast, in some cases like in software product line development, the testability requirements and possibility for conformance testing are emphasized [20].

Software development methods are geared towards producing quality in software products [7]. For example, international standards like ISO 25010 [12] define quality as an amalgam of eight attributes like reliability, operability or security. In addition to these definitions, real-life measurements like the mean time between failures [9] or number of errors found in testing versus errors found after release [25] may also be used as indicators for software development quality.

Organizational testing practices may also vary because of other aspects, such as the development method, resources and customer obligations [14, 17, 32]. Even if the purpose of testing is to verify functionality and to increase product quality [7], practical applications do vary, as different approaches to software development allow different types of tests in different phases. For example, developing software with agile development methods differs from the traditional plan-driven approach to the degree that they can be seen as exclusionary to each other [2]. On the other hand, several techniques like pair programming [10], code reviews [22], think-aloud testing [23, 30] or explorative testing [4] have been developed to enhance product quality and ultimately make the testing process easier. Even the task of generating test cases from which the selection is made varies; for example, black box testing and white box testing define two approaches to case generation based on knowledge regarding the structure of the object being tested [34]. However, these types of approaches focus on the generation process itself, not actually on defining how the test cases are selected, and in case of resource shortages, prioritized.

Overall, there seems to be an abundance of information and studies regarding test case selection in regression testing [e.g. 3, 4, 28], with several different models for cost/benefit-calculations and usability assessment methods. However, there seems to be lack of studies in software development, where regression and conformance testing models are not applicable.

3. RESEARCH METHODS

Software testing is a complex phenomenon, which has several related concepts and different approaches even with seemingly similar organizations [17]. Acknowledging this, we decided to pursue empirical qualitative analysis by applying the grounded theory method [8, 31]. Grounded theory was considered suitable for discovering the basis of testing activities, as it observes and describes real-life phenomena within their social and organizational context. According to Seaman [29], a grounded approach enables the identification of new theories and concepts, making it a valid choice for software engineering research, and consequently, appropriate to our research.

Our approach was in accordance with the grounded theory research method introduced by Glaser and Strauss [8] and later extended by Strauss and Corbin [31]. On the process of building a theory from case study research, we followed guidelines as described by Eisenhardt [5]. The interpretation of field study results was completed in accordance with principles derived from [19] and [24].

3.1 Defining measurements

The ISO/IEC 15504-1 standard [11] specifies an organizational unit (OU) as a part of an organization that deploys one process or has a coherent process context, and operates within a set of business goals and policies. An OU typically consists one part of a larger organization like one development team or regional unit, but small organizations may entirely exist as one OU. In other specifications which are based on ISO15504 [1], like TMMi2 [34], the relation between an organizational unit and rest of the organization is elaborated to allow overlying structures, like the upper management in the company, some steering activities, like policy control over the OU. However, the organizational unit remains a separate actor that operates by an internal process, being responsible for completing the task it has been assigned to, while complying with the policies set by upper organizations. The reason for using an OU as an assessment unit is that this way, the company size is normalized, making direct comparison between different types of companies possible.
In this study, the population consisted of OUs from small, nationally operating companies to large internationally operating corporations, covering different types of software organizations from hardware producers to software houses and to contract testing and consulting services.

### 3.2 Data collection

The initial population and population criteria were decided based on prior research made by our research group [15-17, 32]. We carried out three interview rounds in our study (Table 1). The sample of the first and third interview round consisted of our focus group of 12 OUs collected from our research partners, and later supplemented by researchers to achieve a heterogeneous, polar type sample [5]. The second round of interviews was conducted as a survey with 31 OUs, including the focus group from the first round. Overall, the interviews were done during the winter of 2008-2009.

The 12 OUs in the focus group were professional software producers of a high technical level, with software development as their main activity. The selection of the focus group was based on the polar type selection [5] to cover different types of organizations. The focus group included different business domains and different sizes of companies. The organizations varied (Table 2) from software service consultants to software product developers, extending even to large hardware manufacturers, developing software for their own hardware products. The smallest OU in the focus group was a software developer with approximately twenty full-time employees; the largest was part of an internationally operating software producer employing over 10000 people.

The objective of this approach was to gain a broader understanding of the practice of and to identify general factors that affect test case selection and prioritization. To achieve this, our research team developed two questionnaires and a survey that included questions on themes such as development methods, test processes, test phases, test tools, test automation and quality characteristics. The complete questionnaires and the survey form are available at http://www2.it.lut.fi/project/MASTO/. A reference list of the different themes in different data collection rounds is also available in Table 1.

The interviews contained semi-structured questions, and the whole sessions were tape-recorded for qualitative analysis and to further elaborate on different concepts during the latter rounds. Typically, an interview lasted for approximately one hour and they were arranged as face-to-face interviews with one organization participant and one or two researchers.

The decision to interview designers during the first round was **Table 1. Interview rounds and themes**

<table>
<thead>
<tr>
<th>Round type</th>
<th>Number of interviews</th>
<th>Interviewee role</th>
<th>Description</th>
<th>Themes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Semi-structured</td>
<td>12 focus OU interviews</td>
<td>Designer or Programmer</td>
<td>The interviewee was responsible for or had influence on software design.</td>
<td>Design and development methods, Testing strategy and methods, Agile methods, Standards, Outsourcing, Perverced quality</td>
</tr>
<tr>
<td>2) Structured with Semi-structured</td>
<td>31 OUs, including 12 focus OUs</td>
<td>Project- or Testing manager</td>
<td>The interviewee was responsible for the software project or testing phase of the software product.</td>
<td>Test processes and tools, Customer participation, Quality and Customer, Software Quality, Testing methods and resources</td>
</tr>
<tr>
<td>3) Semi-structured</td>
<td>12 focus OU interviews</td>
<td>Tester or Programmer</td>
<td>The interviewee was a dedicated tester or was responsible for testing the software product.</td>
<td>Testing methods, Testing strategy and resources, Agile methods, Standards, Outsourcing, Test automation and services, Test tools, Perceived quality, Customer in testing</td>
</tr>
</tbody>
</table>

**Table 2- Description of the Interviewed OUs**

<table>
<thead>
<tr>
<th>OU</th>
<th>Business</th>
<th>Company size / Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case A</td>
<td>MES(^1) producer and electronics manufacturer</td>
<td>Small / National</td>
</tr>
<tr>
<td>Case B</td>
<td>Logistics software developer</td>
<td>Large / National</td>
</tr>
<tr>
<td>Case C</td>
<td>ICT consultant</td>
<td>Small / National</td>
</tr>
<tr>
<td>Case D</td>
<td>Internet service developer and consultant</td>
<td>Small / National</td>
</tr>
<tr>
<td>Case E</td>
<td>Safety and logistics system developer</td>
<td>Medium / International</td>
</tr>
<tr>
<td>Case F</td>
<td>Financial software developer</td>
<td>Medium / National</td>
</tr>
<tr>
<td>Case G</td>
<td>ICT developer and consultant</td>
<td>Large / International</td>
</tr>
<tr>
<td>Case H</td>
<td>Financial software developer</td>
<td>Large / International</td>
</tr>
<tr>
<td>Case J</td>
<td>SME(^2) business and agriculture ICT service provider</td>
<td>Small / National</td>
</tr>
<tr>
<td>Case K</td>
<td>MES(^1) producer and logistics service systems provider</td>
<td>Medium / International</td>
</tr>
<tr>
<td>Case L</td>
<td>Modeling software developer</td>
<td>Large / International</td>
</tr>
<tr>
<td>19 survey-only cases</td>
<td>Varies; from software consultancies to software product developers and hardware manufacturers.</td>
<td>Varies</td>
</tr>
</tbody>
</table>

\(^{1}\)Manufacturing Execution System  \(^{2}\)SME definition [6]
based on our aim to gain a better understanding of the operational level of software development. We wanted to see whether our hypotheses from our prior studies [15-17, 32] and literature review were valid. The interviewees in the first round were selected from a group of developers or programmers, who had the possibility to decide on or affect the structure of the software product. In one first-round interview, the organization interviewed was allowed to send two interviewees, as they considered that the desired role was a combination of two positions in their organization. In another first-round interview, we allowed the organization to supplement their answers, as the interviewee considered that the answers lacked some relevant details.

For the second round and the survey, the population was expanded by inserting additional OUs to enable statistical comparison of results. Selecting the sample was demanding because comparability was not specified by a company or an organization but by an OU with comparable processes. With the help of authorities (the network of the Technology and Innovation Agency of Finland) we collected a population of 85 companies. Only one OU from each company was accepted to the population to avoid bias of over-weighting large companies. From this list, the additional OUs accepted to the survey sample were selected according to the population criteria used in the first interview round.

We expanded the sample size in the second round to 31 OUs, including the OUs of the first round. The purpose of combining the interviews and the survey was to collect data more efficiently, simultaneously gaining a generalized perspective with survey-sized data, and obtaining detailed information about test management for the grounded analysis.

During the second round of data collection, our decision was to interview and simultaneously conduct a survey where the population consisted of project or test managers. The objective was to collect quantitative data about the software and testing process and further to collect qualitative material about various testing topics, such as test case selection and agile methods in the software process. We selected managers for this round as they tend to have more experience about software projects; they have a better understanding of the overall software testing process and the influence of upper management policies in the OU.

In the third round, the same sample organizations were interviewed as in the first round. The interviewees of the third round were testers, or in the case where the OU did not have separate testers, programmers whose tasks included module testing were interviewed. The interviews in these rounds focused on such topics as problems in testing (such as complexity of the systems, verification, and testability), the use of software components, testing resources, test automation, outsourcing, and customer influence in the test process.

The interview rounds, interviewee roles in the organization and study structure are summarized in Table 1, and the participating organizational units are summarized in Table 2.

### 3.3 Data Analysis

The grounded theory method contains three data analysis steps: open coding, where categories and their related codes are extracted from the data; axial coding, where connections between the categories and codes are identified; and selective coding, where the core category is identified and described [31].

The objective of the open coding was to classify the data into categories and identify leads in the data. The process started with “seed categories” [5] that contained essential stakeholders and known phenomena based on the literature. Seaman [29] notes that the initial set of codes (seed categories) comes from the goals of the study, the research questions, and predefined variables of interest. In our case, the seed categories were derived and further developed based on our prior studies on software testing, and from the literature. These seed categories were also used to define themes for the questions in the questionnaire, including topics such as development process, test processes, testing tools, automation or role of the customer. A complete list of the seed categories, and general themes of the study, is in Table 1.

In open coding, the classified observations are also organized into larger categories. New categories appear and are merged because of new information that surfaces during the coding. For example, our initial concept of having quality as a separate category was revised and quality was included within other categories such as criticality or outsourcing as an attribute with an “effect on quality”. Another notable difference from the seed categories was that the management and policies were not as restrictive as originally thought, so they were incorporated into such themes as project management and test planning. Additionally, concepts like process difficulties or improvement proposals were given their own categories. At the end of the open coding, the number of codes was in total 166 codes, grouped into 12 categories.

The objective of the axial coding was to further develop separate categories by looking for causal conditions or any kinds of connections between the categories. In this phase, the categories and their related observations were becoming rigid, allowing the analysis to focus on developing the relationships between larger concepts. In this phase, the categories formed groups in the sense that similar observations were connected to each other. For

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applied selection approach</td>
<td>The method the organization is currently using to select which test cases are included in the test plan.</td>
</tr>
<tr>
<td>Software type</td>
<td>The type of software the OU is developing.</td>
</tr>
<tr>
<td>Test designers</td>
<td>The personnel responsible for designing and selecting the test cases.</td>
</tr>
<tr>
<td>Development approach</td>
<td>The method the organization is currently using to develop software.</td>
</tr>
<tr>
<td>Testing resources</td>
<td>An approximation on how large an amount of testing resources the organization currently has access to, in comparison to the optimal, i.e. perfect amount of resources.</td>
</tr>
<tr>
<td>Customer influence</td>
<td>The type and method of customers to influence the organization’s software test process.</td>
</tr>
<tr>
<td>Selection problem</td>
<td>The most common process hindrance the test case selection method causes to the organization.</td>
</tr>
<tr>
<td>Explorative testing</td>
<td>Does the organization apply non-predefined test cases in their test plan?</td>
</tr>
</tbody>
</table>
example, codes such as “Process problem: outsourcing”, “Outsourcing: Effect to quality” and “Development process: support for outsourced activities” formed a chain of evidence for observing how the outsourced resources in development fitted in to the overall process. By following this type of leads in the data, the categories were coded and given relationships with each other.

The third phase of grounded analysis, selective coding, was used to identify the core category [31] and relate it systematically to the other categories. As based on [31], the core category is sometimes one of the existing categories, and at other times no single category is broad or influential enough to cover the central phenomenon. In this study, the examination of the core category resulted in a category applied case selection approach, with a set of software testing concepts listing issues related to the core category or explaining the rationale for observed activities. The core category was formed by abstracting the categories and defining a common denominator, because none of the categories was considered influential enough to explain the entire phenomena. For example, we observed the primary case selection method in all of our organizations, but were unable to define one cause for the approach the organizations applied. Our initial approximation that the case selection method was closely connected to the development method and the role of the customer was partially correct, but we also identified several other aspects like amount of resources or test case developers, which also seemed relevant. Overall, we adjusted the core category to include all these concepts, which also became the categories presented in this paper. Additionally, by identifying the core category and affecting factors, we were able to define and name two approaches for selecting the approach for test case selection.

4. RESULTS AND OBSERVATIONS

In the following section we present and discuss the observations from our study. First of all, we were able to identify several concepts which would affect the case selection method, and introduce them in the first part. Secondly, we elaborated the observation made from the categories into hypotheses, which summarize and explain how organizations in this study selected test cases. Finally, in the third part we introduce our two stereotypes of selection methods.

4.1 Developed categories

The categories were developed based on their observed effect on actual test case selection and their ability to interpret why the organization had decided to use this approach. These categories were all related to the core category, identified during selective coding, and had a definite impact on how the organization approached test case selection or explained the differences between organizations. For example, the category applied selection method was taken directly from the observation data as it discussed the studied phenomenon of case selection, while software type and development approach were used to establish the objectives and operating methods of the software development organization. The category selection problem was also taken directly from observations as it discussed the difficulties in applying the used approach. The categories of test designers, testing resources, customer influence and explorative testing were included as they were observed to follow a pattern based on case selection method. The complete list and short summary of the developed categories are available in Table 3.

The category of applied selection method describes the primary way the organization selects what features or use cases are tested during development. Selection seems to be based on one of two major approaches: the risk-based “Which causes the largest expenses if it is broken?” and definition-based “Which are the main functionalities the software is supposed to do?”. In some organizations, there are also some secondary concerns like conformance testing to ensure that the system complies with some interface or with a set of established requirements, or “changes first”, where the most recent changes have priority over other test cases.

The category software type defines the type of software the organization is building as their main product. In this study, the development outcomes were classified into three categories: software service, software product and software module for hardware. In software service, the software is used as a network-based application or front-end for network service, including Internet services. In software product, the application is stand-alone software installed on a platform such as a PC or mobile phone. The last category, software module for hardware refers to embedded software for dedicated devices.

The category test designers defines the personnel responsible for defining and designing test cases or authorized to decide on what area the testing effort is focused on. In several organizations, the test cases are designed by the programmers themselves or by designated software structure designers. The management level, made up of test managers or project managers, was responsible for designing test cases in five organizations, and the clients were allowed to define test cases in three organizations. Overall, the responsibility for test designing varied between organizations.

The category of development approach defines the approach the organization applies to software production. This category is defined based on a linear dimension defined by Boehm [2], where the polar points represent fully plan-driven and fully agile development, with overlap in the middle, combining techniques from both sides. For example, several organizations adopt only some activities from agile methods and apply them in the traditionally plan-driven environment, or apply the agile approach to smaller support projects, applying plan-driven methods to the main product development projects.

The category of testing resources is an indicator of how many resources the test organization has when compared to their optimal, i.e. perfect situation. In this category, we apply a scale with three possibilities, Low (33% or less), Moderate (34-66%) and High (67% or more). For example, if an organization currently has two dedicated testers and thinks that they could use three, it would mean a resource availability of 67 %, translating to “High” on the scale. It should be noted that in this scale, the score less than “High” does not necessary mean that the test process is inefficient; the scale is merely an indicator of the amount of resources allocated to testing. The ratings, presented in the Table 4, are based on the answers given by the organization during the second round survey.

The category of customer influence defines the part customers have in the development process. The most common ways of influencing a project was by directly participating in some testing phase, by approving the test results or approving the test plan made by the developer organization.
The category of selection problem defines the process hindrances caused by the test case selection approach. In risk-based selection, the common hindrances were that the test cases either did not cover all of the important cases or that the designed cases were discarded from the final test plan. With the design-based approach, the problems were usually at the management level, being caused by such concepts as restrictive test policies or managing test process to meet all required and formal activities, like communications, paperwork, schedules, test environments, weekly reviews, project steering group meetings and such; In layman’s terms, the increased amount of red tape.

Finally, the category of explorative testing indicates whether or not the organization applies explorative testing methods. For this category, all testing methods which apply non-predefined test types, like interface or usability-testing, were considered explorative. In this category, the organizations were strongly divided into two opposing groups; some organizations considered explorative testing as an important phase where usability and user interface issues were addressed, whereas some organizations considered testing without test cases and documentation as a waste of test resources.

### 4.2 Hypotheses and Observations

Our study developed hypotheses based on the observations regarding test case selection. The hypotheses were shaped according to the categorized observations listed in Table 4, by developing concepts that explained the observations and followed the rational chain of evidence in the collected data. For example, the first hypothesis was generalized from the observation that all organizations which applied a design-based approach also favored plan-driven product development, and that their customers tended to have influence on the design-phase of the product. Following this lead, we focused on these observations and tried to define exactly how the risk-based approaches differed in the design-phase and would this observation be generalizable enough for creating a hypothesis. A similar approach was used with hypotheses two and three. The last hypothesis, number four, came from the general observation that for some reason, several organizations considered explorative testing to be wasteful or a futile waste of resources, whereas some thought that it was one of the most important aspects in testing. As this behavior was not as

<table>
<thead>
<tr>
<th>Case</th>
<th>Applied selection method</th>
<th>Software type</th>
<th>Test designers</th>
<th>Development approach</th>
<th>Testing resources</th>
<th>Customer influence</th>
<th>Test case selection problem</th>
<th>Explorative testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Risk-based with changes first</td>
<td>Software module for hardware</td>
<td>Programmers</td>
<td>Plan-driven supported by agile</td>
<td>Low</td>
<td>Approves product</td>
<td>Important test cases are discarded</td>
<td>Yes, programmers do it.</td>
</tr>
<tr>
<td>B</td>
<td>Risk-based with changes first</td>
<td>Software product</td>
<td>Designers</td>
<td>Agile</td>
<td>Moderate</td>
<td>Participates on testing</td>
<td>Agile products seem to be difficult to test.</td>
<td>No, only defined cases are tested.</td>
</tr>
</tbody>
</table>
| C    | Risk-based with changes first | Software product | Programmers with clients | Agile | Moderate | Participates on testing | Some test cases are not implemented. | Yes, programmers do it.
| D    | Risk-based | Software service | Programmers | Plan-driven supported by agile | Low | Approves testing plan | Some test cases are not implemented | Yes |
| E    | Risk-based | Software module for hardware | Programmers | Agile supported by plan-driven | High | Approves product | Important test cases are discarded | Yes, some phases apply. |
| F    | Risk-based with conformance | Software module for hardware | Designers | Plan-driven | Moderate | Approves product | Some test cases are not implemented | Yes |
| G    | Design-based with conformance | Software service | Test manager with testers | Plan-driven | High | Approves testing plan | Validating functionalities is difficult | No, only defined cases are tested. |
| H    | Design-based with conformance | Software service | Designers with testers | Plan-driven | High | Approves testing plan | Amount of policies affect test effectiveness | No, not enough time. |
| I    | Design-based with conformance | Software service | Test manager with testers | Plan-driven | High | Approves design | Too large reliance on test manager experience | No |
| J    | Risk-based, changes first | Software product | Project manager | Plan-driven supported by agile | High | Participates on testing | Important test cases are discarded | Yes |
| K    | Design-based | Software module for hardware | Project manager, clients | Plan-driven supported by agile | Moderate | Participates on test design | Some test cases are not implemented | Yes, in some projects. |
| L    | Design-based | Software product | Project manager with designers | Plan-driven | High | Approves product | Test management in large projects | Yes, several phases apply. |

Table 4. Observations on test case selection method
systematic with our observations as some other aspects of test case selection, it was included as a separate observation, and subsequently a separate hypothesis, on test case design and case selection.

Hypothesis 1: Risk-based selection is applied when the software design is not fixed at the design phase. Risk-based selection was used in all those organizations that applied primarily agile development methods in their software process. Furthermore, all organizations that applied traditional plan-driven software development methods also applied the design-based test case selection approach. With the risk-based approach, the selection was clearly based on communication between case selectors and stakeholders:

“Basically our case selection method is quite reactive [to feedback].” –Case E, Tester

“I might use risk-based techniques based on the advice from developers.” – Case B, Designer

In the design-based approach, software process management gets much more involved:

“Test manager decides based on the requirements on what will be tested.” – Case G, Tester

“Designers with project manager decide on the test cases.” –Case I, Designer

In general, it also seemed that in the organizations that applied the risk-based approach, customers had a big influence on the latter parts of the software process, either by approving the final product or by directly participating in the latter test phases.

“...so far we have been able to go by trusting [final] testing phases to the customer.” – Case C, Designer

“For larger projects we give our product to a larger client for test run and see how it works.” – Case A, Tester

In organizations applying the design-based approach, the customer input in test design was more indirect, including approaches like offering supplemental test cases or reviewing case selections.

“...customers can come to give us their test case designs so we can accommodate to their requirements.” –Case K, Designer

“Customer usually gives input on test design if the test plan has shortcomings or is overly vague.” – Case H, Tester

Hypothesis 2: The design-based approach is favored in organizations with ample resources and but it requires more management. The most definite difference between organizations that chose the design-based approach was that most of them reported a high amount of testing resources. On average, companies with the design-based approach had 73% of required resources, while in the risk-based group, the average was 49%.

Another indicator of the differences between the two groups was the types of problems the testing process experienced in their case selection. In the risk-based selection, the most common process difficulty was related to the test cases. Either they did not cover all the critical cases, or they discarded critical cases from the final test plan.

“The problem is in defining what should be tested.” –Case A, Designer

“The document quality fluctuates between projects... sometimes the critical test cases should be defined more clearly.” – Case C, Designer

“What we truly miss is the ability to test all modules consistently.” – Case D, Designer

In the design-based approach, the most common problems were related to managing the testing process, satisfying testing criteria set by test policies or keeping up with the requirements.

“It is up to test managers and their insight to define a satisfying test case design.” – Case I, Designer

“We are already at the full capacity with test cases, we should start discarding some of them...” – Case K, Tester

“[Policy makers] really cannot put testing into a realistic timetable.” – Case H, Tester

An interesting observation was that in the design-based approach, the test cases were mostly designed by separate test process management or test managers, whereas in the risk-based approach the design was done by software developers: programmers or software designers.

Hypothesis 3: The use of test automation is not affected by the case design or case selection approach. The effect of test case selection approaches on feasibility or applicability of test automation was also examined, but the results did not yield any relevant information or distinct pattern. Aside from prior observations on test automation [15], the decision of applying test automation did not seem to be connected to test case selection, meaning that the decision to implement automation is based on other test process factors. For example, Case B from the risk-based group was an active user of test automation services:

“All our projects have test automation at one time or another.” – Case B, Designer

In the design-based approach, Cases G and K had a significant number of automated test cases in their software development process:

“...for some ten years most of our conformance test cases have been automated.” – Case G, Tester

“Well, we have test cases which are automatically tested during the nighttime for every daily build.” –Case K, Tester

In fact, all of the organizations had some forms of automation, were introducing automation to their test process or could see some viable way of applying test automation.

“Regression tests which are build on our own macro-language, and some unit tests.” – Case G, Designer

“[new testing tool] is going to allow automation.” – Case A, Manager

“We are implementing one for interface testing” – Case I, Tester

Hypothesis 4: Explorative testing may be seen by policy makers as an unproductive task because of its ad hoc nature. The explorative testing methods in this study included all test methods
and practices where testers did non-predefined test activities as a part of the standard test process. In organizations where the risk-based approach was applied, explorative testing was commonly applied, whereas in the design-based approach the amount of exploration was noticeably smaller.

“[programmers] are allowed to do tests as they please” – Case A, Designer

“Yes we do that, however the benefit of that work varies greatly between individuals.” – Case E, Tester

“Those ‘dumb tests’ really bring up issues that escaped developers designs.” – Case F, Tester

However, by comparing the organizations based on their originating company sizes, it becomes evident that large-sized companies used less explorative test methods. One reason for this could be that explorative testing is difficult to document; in other words the explorative test process would cause additional requirements for management and policies.

“We have so much other things to do...no time for that [explorative testing]” – Case H, Tester

“It would be interesting but no, we do not do that kind of thing.” – Case I, Tester

“Well maybe if there were some unusual circumstances but I think no; even in that case we would probably first make plans.” – Case G, Tester

### 4.3 Observed approaches

Based on the observations above, we are able to conclude that the software case selection approaches tend to resemble two basic approaches: risk-based and design-based selection. Typically in the risk-based approach, test design tasks are planned and completed by software developers, whereas in the design-based approach, management and separate test managers are responsible for the case generation. It seemed that the organizations applying risk-based approaches were also more likely to apply agile methods in their software processes. However, also some design-based organizations applied agile methods if it was deemed necessary. This behavior could be explained with customer participation. As the risk-based approach also favors customer participation in the latter parts of the process, it allows a customer to request last-minute changes. As agile development does not create a strong, “iron-bound” design for the software product, but rather general guidelines for development objectives, it would also seem reasonable to assume that test cases are selected based on foreseeable risks and not based on the design documentation which may lack details. These two approaches are summarized in Table 5.

The selection of the risk-based approach was also favored when the testing resources were limited. If testing is organized with limited resources, the prioritization of test cases takes place, favoring the risk-based approach. In this situation, the obvious choice is to allocate resources to address the most costly errors. There are studies showing that by prioritizing test cases, the test process can be organized effectively with as low as 15% of the desired resources [25]. The costs caused by product defects offer an easy and straightforward measurement method to determine which cases should be tested and which discarded as an acceptable expense.

Besides serving as the method of test case prioritization, the risk-based approach was also more likely to supplement test cases with exploratory testing practices, a phenomenon that may be related to the test policy issues of the design-based approach. Where the design-based approach was applied, the organizations emphasized management and policies. The actual type of software product seemed to have little to no impact on selection approach.

### 5. DISCUSSION

As software testing usually only has a limited amount of resources [18], there always has to be some form of selection process on which parts of the software should be tested and which can be left as they are. Petschenik [25] discusses this phenomenon by implying that the testing process can be organized effectively with merely 15% of the required resources; Huang and Boehm [9] indicated that a 20% investment can cover 80% of the testing process if the test case design and test focus is selected correctly. In practice, we observed the same phenomena, as several organizations reported a resource availability of 60-70%, indicating that they do prioritize with their test cases.

Our study examined how test cases were designed and selected for test plans in 12 professional software organizations. The results indicate that test case selection seems to generalize into two approaches; risk-based and design-based. In the risk-based approach, test cases are selected on the basis that most costly errors are eliminated from the software. In many cases it is the economically preferable strategy to keep deadlines rather than to extend testing phases [9,27]. In these cases, testing resources are more likely geared towards minimizing the costs caused by errors found after the release.

The other selection method is the design-based approach. In this

<table>
<thead>
<tr>
<th>Category</th>
<th>Risk-based selection</th>
<th>Design-based selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test designers</td>
<td>Developers: programmers and testers</td>
<td>Managers: test and project managers</td>
</tr>
<tr>
<td>Development approach</td>
<td>Leans towards agile methods</td>
<td>Leans towards plan-driven methods</td>
</tr>
<tr>
<td>Testing resources</td>
<td>Limited</td>
<td>Sufficient</td>
</tr>
<tr>
<td>Explorative testing</td>
<td>Applied commonly</td>
<td>Applied rarely</td>
</tr>
<tr>
<td>Effect of policies in decisions on testing.</td>
<td>Small; most decisions done in project level.</td>
<td>Large; most decisions are based on company policies or customer requirements.</td>
</tr>
<tr>
<td>Customer influence</td>
<td>In the testing process</td>
<td>In the design process</td>
</tr>
<tr>
<td>Limitations of the model</td>
<td>Test case coverage may become limited.</td>
<td>Test process may become laborous to manage</td>
</tr>
<tr>
<td>Design concept</td>
<td>“What should be tested to ensure smallest losses if the product is faulty?”</td>
<td>“What should be tested to ensure that the product does what it is intended to do?”</td>
</tr>
</tbody>
</table>
approach, the organization decides the test cases based on the design documentation of the product, ensuring that the software is capable of performing the tasks it is supposed to do. The design-based approach seems to be favored in organizations that have sufficient or ample testing resources. These organizations may also have stricter customer-based or policy-defined activities in their software process, like following a strict formal process, or requiring customers to approve all decisions which affect the project. The most common process hindrances in the design-based approach seem to be policy restrictions and management issues like rigid processes, top-heavy management and communicating between all relevant stakeholders.

The most obvious limitation of this study is the number of organizations. Our study interviewed 36 software professionals from 12 different organizations, which were selected to represent different types and sizes of software organizations. For this type of study, Onwuegbuzie and Leech [37] discuss the several threats associated to the validity. In their opinion, internal validity and external credibility should be maintained by providing enough documentation, explaining the applied research method and providing proof of the chain of evidence that led to the study results. In this project, the internal validity was maintained with these viewpoints in mind. For example, we applied several researchers in designing the questionnaires, and later the same researchers collected, and subsequently analyzed the data. In addition, we conducted a survey in 31 organizations to collect quantitative data to compare and cross-reference our qualitative observations with quantitative data.

The objective of this qualitative study was not to establish statistical relevance, but to observe and explain the strategies of how real-life organizations decide which test cases to select. Our analysis revealed two selection approaches with several characterizing attributes explaining the differences. However, they also shared some attributes like software types, so in practice they more likely complement each other and the division is not as straightforward as it may seem based on the results.

6. CONCLUSIONS
In the observed organizations, test cases were selected using two main approaches: the risk-based and the design-based approach. Generally, in organizations where testing resources were limited and the product design was allowed to adapt or change during the process, the risk-based approach became increasingly favored. When the project was allowed more testing resources and the software design was made in a plan-driven fashion, the objective for the test process shifted towards test case coverage, and subsequently, towards the design-based approach in test case selection. In these cases, the case selection was based on product design and the verification of features, not in damage prevention and minimizing the possible risks. However, in practice the shift between approaches was not as clear-cut as it may seem; additional concepts like policies, customers and development methods can also affect the selection.

In this paper, we observed and present results on how software test cases are selected and how test plans are constructed with different amounts of resources in different types of software organizations. We believe that software organizations can achieve better productivity by defining the test process and by focusing on the critical aspects for test process to validate. By designing the test cases to more closely fit the needs of the organization and product characteristics, test process issues can be better addressed and more attention can be given to the aspects that need enhancement. Therefore, these results can be used to develop testing practices and generally to promote the importance of designing test plans to fit the process organization.

7. ACKNOWLEDGMENTS
This study was supported by the ESPA project (http://www.soberit.fut.es/spa/), funded by the Finnish Funding Agency for Technology and Innovation, and by the companies mentioned at the project web site.

8. REFERENCES