Test Overlay in an Emerging Software Product Line – An Industrial Case Study

Engström, Emelie; Runeson, Per

Published in:
Information and Software Technology

DOI:
10.1016/j.infsof.2012.04.009

2013

Test Overlay in an Emerging Software Product Line – An Industrial Case Study

Emelie Engström and Per Runeson

Abstract

Context: In large software organizations with a product line development approach, system test planning and scope selection is a complex task. Due to repeated testing: across different testing levels, over time (test for regression) as well as of different variants, the risk of redundant testing is large as well as the risk of overlooking important tests, hidden by the huge amount of possible tests. **Aims:** This study assesses the amount and type of overlaid manual testing across feature, integration and system test in such context, it explores the causes of potential redundancy and elaborates on how to provide decision support in terms of visualization for the purpose of avoiding redundancy. **Method:** An in-depth case study was launched including both qualitative and quantitative observations. **Results:** A high degree of test overlay is identified originating from distributed test responsibilities, poor documentation and structure of test cases, parallel work and insufficient delta analysis. The amount of test overlay depends on which level of abstraction is studied. **Conclusions:** Avoiding redundancy requires tool support, e.g. visualization of test design coverage, test execution progress, priorities of coverage items as well as visualized priorities of variants to support test case selection.

1. Introduction

In large software organizations with a product line development approach, selective testing of product variants is necessary in order to keep pace with the available time to market for new products. The number of testing combinations in such variability-intensive contexts is extensive. Testing is repeated across three dimensions [15]: 1) the traditional testing at different levels of abstraction (unit, integration, system etc.), 2) regression testing as the system evolves over time, and 3) testing over different product variants, sharing a common code base, see Figure 1. This entails a high risk of redundancy in the testing and also the reverse, that important aspects of the differences between the tested versions and variants are overlooked. One of the major challenges in testing a software product line (SPL) is the balancing of testing efforts across these three dimensions [7].

To handle the combinatorial explosion in software product line development, regression testing approaches are suggested to be applied not only to versions

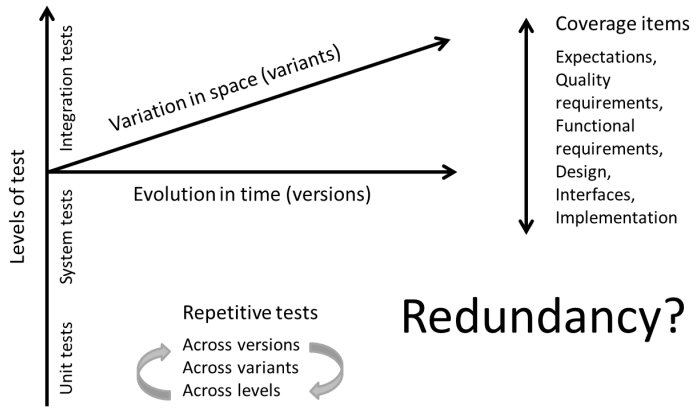

Figure 1: Testing in a product line context is repeated across three dimensions [15]: testing at different levels (e.g. unit tests, integration tests and system tests), testing across versions (e.g. the continuous software updates), testing of multiple variants (e.g. adaptations to different hardwares). Testing at different levels covers the system from different perspectives (e.g. expectations, different types and detail levels of requirements, design or code)

.

but also to variants [7]. Regression test selection strategies aim at optimizing the testing after a change by focusing the testing on parts that may have been affected by a change. We study the amount of test *overlay* (i.e. multiple tests of the same entity) and the extent to which it is *redundant* (i.e. one could be replaced by the other) in a large-scale real life software product line development context.

An in-depth case study was launched including both qualitative and quantitative observations. Recent systematic literature reviews on regression testing have indicated the lack of industrial case studies [9, 22]. Most studies are done in the small, and hence questions of scalability and usability are left unanswered [9]. We aimed to bridge the gap between research and practice, for which a better understanding of the real-life context is needed, and thus we launched a case study [16].

The studied case is the testing in the case company's development of Android embedded devices. For the in-depth analysis, the testing of one function is studied. The function exists in several product variants, depends on hardware variants, evolves in different versions over time, and is adapted to continuous upgrades of the Android platform. The development organization is complex, involving a globally distributed development, and the market situation involves delivery of tailored product variants to customers, based on varying market and country specifications. From a business perspective a product line strategy is in place. However, the technical and methodological benefits of the systematic

reuse are not yet fully exploited, hence we call it an emerging product line.

The focus in this study is on manual testing of functional and quality requirements, since this is the most personnel resource demanding testing. Manual testing is characterized by a higher degree of creativity for the tester and less detailed documentation, although some general principles on test redundancy are shared with automated testing. Relations between different test executions and implicit testing goals are identified and expressed in terms of coverage items (CovI:s), capturing different levels of abstraction as well as different purposes of tests. For the assessment of coverage and overlay, a method for visualization of test execution progress is proposed and illustrated.

To our knowledge, no exploratory case studies of test overlay in a variability-intensive context have been reported earlier. However, the complexity of the issue is well known and is discussed from many different perspectives: software product line testing [7], testing incrementally evolving software [11], testing of component based systems [21], testing of web based applications [2] and of service oriented architectures [4], as well as from a configuration management perspective [3, 19].

The article is organized as follows. Section 2 describes the design of the case study, including the theoretical framework, the case study context and methods used. Analysis criteria for the quantitative assessment of overlay are introduced in Section 3 and the quantitative results are presented in section 4. Quantitative and qualitative results are analyzed in Section 5 (Existence and causes of redundancy) and Section 6 (Visualization). Section 7 concludes the findings of the study.

2. Case study design

The design of this case study is outlined below in line with the guidelines by Runeson *et al.* [16], and contains accordingly:

- the rationale for the study,
- the objectives and research questions,
- the case study context,
- a description of the case and its units of analysis,
- the theoretical frame of reference for the study
- propositions
- concepts studied and related measures
- procedures for the case study
- methods for data collection and analysis, and
- issues related to the validity of the study.

These items are presented in the following subsections.

2.1. Rationale

This work continues our research on regression testing and software product line testing. We have reviewed the research on regression test selection [9] and product line testing [7], conducted a survey on industrial practices [6], and applied regression test selection procedures in-the-small [10, 8].

Several challenges for testing a software product line have been brought up by researchers, one of the most urgent is how to handle the huge amount of possible tests [7]. A common proposal for how to support test planning and test selection in such variability-intensive context, is the application of regression testing strategies to variants of the software, as well as to versions [18].

Even though regression testing has been researched to a large extent [9, 22], the application of research outcomes to software engineering practice is not easily done. The gap between research and practice is large, the evidence base is inconsistent and most techniques for regression test selection are evaluated off-line in small scale experiments, hence questions of scalability and usability are not researched [9]. In order to enable bridging the gap between research and practice, a better understanding of the real-life context is needed, which is the motivation for our choice of the case study methodology.

Our focus on *test overlay* is motivated by the underlying assumptions of most regression testing techniques, i.e. it is possible to predict which test cases are most likely to trigger failures that help detecting faults based on available information on, for example, changes and test execution history. The regression tests focus on changed parts and potential side effects, assuming that previous test results are only valid for reuse if not related to these categories. This implies that a subset of the test cases is redundant, and that testing would be more efficient if guided by an analysis of the differences between the system under test and the previously tested version or variant of the system.

2.2. Objective

Our objective is to investigate the phenomenon of "test overlay" in a large-scale product line context, for the purpose of gaining a better understanding of the potential for selective testing approaches in this context and identification of how to guide test planning and selection based on regression testing concepts. Three main questions are investigated:

RQ1 *How much testing in a variability-intensive context is overlaid, and which is redundant?* – Testing is repeated across abstraction levels, evolution over time (versions) and over space (variants) which could imply that multiple testing is done on the same items. How much of the overlaid testing is really redundant?

RQ2 *When and why does overlaid testing appear?* – If overlaid testing exist, which factors are causing the overlaid testing?

RQ3 *How can visualization support test scoping decisions?* – We assume that visualization is a powerful tool when handling large volumes of data, see for example Zaidman *et al.* [23]. Thus it is relevant to study prerequisites for visualization in this context.

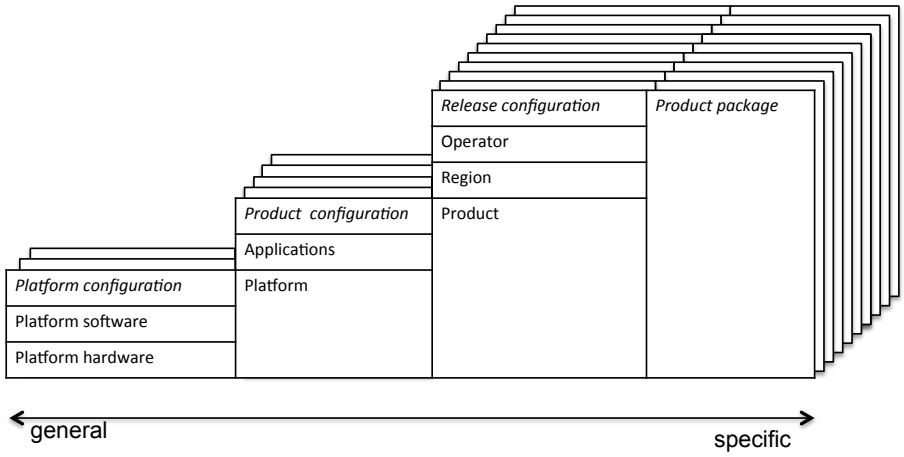

Figure 2: Configuration view of the product line under study.

The first research question is mostly *descriptive*, the second is *explanatory*, while the third question comprises an *improving* component [16].

2.3. Context

This section presents the study context, as much as we can do for confidentiality reasons. The case study is conducted at a company developing mobile devices with embedded real-time software in a domain which is very competitive both regarding quality and innovation. The development process is highly iterative, and the same software basis is used in several product versions and variants. The facets of the context is described below according to the framework proposed by Petersen and Wohlin [13]. The terminology in the context description is changed to align to generally accepted definitions, if needed.

2.3.1. Products and market

The products under study are mobile devices with embedded real-time software. The products are based on the Android platform, which in itself comprises more than 10 million lines of code. The platform is adapted and extended to comprise more and specialized features, and to fit the specific hardware of the device. Instances of specific combinations of hardware and software platforms are referred to as *platform configurations* in Figure 2.

The product line products, developed in a software project, comprise different product variants (about 10), called *product configurations* in Figure 2. The products are in turn customized for a number of different customers and market segments (hundreds) which have different software requirements, and are called *release configurations*. Each release is packaged in a *product package*, including physical packaging, defaults settings etc. Several (a handful) projects are ongoing in parallel and software is reused across projects as well.

The quality requirements are traditionally high in the telecom domain, especially regarding performance and reliability [1], and since the market is very competitive and fast changing, on time delivery is crucial.

2.3.2. Organization and process

The development organization is globally distributed, applies incremental development practices, and distributes testing tasks over different organizational units. In more detail:

- *Incremental development* – The software development process is an incremental process, where each feature is developed and integrated in small iterations. Thus there is a need for continuous regression testing during development.
- *Organizational distribution* – The software development is distributed over three organizational units (*core software, application software* and *product composition*, See I in Figure 3), where they primarily focus on platform, product and release configurations, respectively, as defined in Figure 2. Within the core software and application software organizations, the software is divided into different functional areas and for each area there is a team of developers and testers.
- *Global distribution* – Parts of the organizations are distributed over three continents.
- *Test levels* – Testing in each of the core software and application software organizations are conducted at (at least) three main levels of test abstraction, which involves repeated testing of common parts. *Feature testing* (unit testing, structural and functional testing) are carried out by the functional area teams while *Integration testing* and *system testing* are carried out by dedicated test teams. Within the third organization, at the product composition level, all product configurations are tested with system tests and all product packages are acceptance tested. *Regression testing* is conducted at all test levels and organizational units.
- *Test practices* – There is no centralized strategy for neither test case design nor test selection, only guidelines and working practices for organizational units exist. For each new feature, several new test cases are created based on the feature requirements. Test cases are selected based on practitioners' experience.

2.3.3. Tools

All test case descriptions and the execution data are stored in a commercial tool, HP's Quality Center (QC). QC is a web based test database that supports essential aspects of test management. Test planning, test design and test reporting is performed in the QC environment.

Test execution is done both manually and automatically, the latter using a combination of proprietary and in-house tools. However, this study focuses only on the manual test execution.

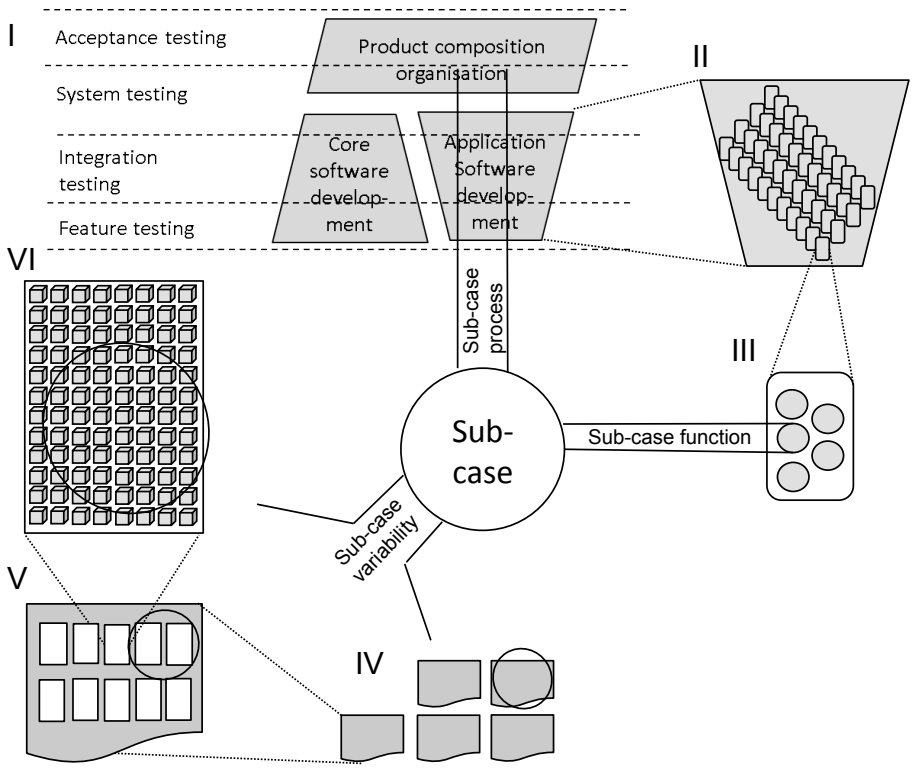

Figure 3: The case covers three organizational units and four test levels (I). The scope of the sub-case is delimited to one single function (III) which is one out of five main functions of a feature area (II). Furthermore the sub-case comprises two organizational units and three levels of test (I) and a share of the variability, i.e. one out of a handful parallel projects (IV); two product variants (V) and about half of the available product configurations (VI).

2.4. Case and units of analysis

This study is a single-case study [16] in the emerging product line development context, see Figure 3, where we define two study contexts, one of which is a subset of the other. The larger context, the *case development context*, has the three development organizations as its main unit of analysis. The sub-context is scoped down to the *sub-case function development context*, which scopes one single function, tested from different perspectives across test levels and organizations.

The function selected for the sub-case context is not aimed to be representative in any means. It was chosen to include functionality which is complex and challenging enough to span over several components and product variants, and cause interaction with several organizational units. This is illustrated in Figure 3. The selected function is one out of five main functions of a feature, which in turn is a bring up to the current project based on an application developed at another site. The size of the feature is about 35.000 LOC.

The main units of analysis are the three main organizational units: *Core software, Application software* and *Product composition*. Each organization conducts testing at several testing levels (feature tests, integration tests, and system tests) and at each test level, several test activities with differences in scope and focus are carried out, see Figure 3. The sub-units of analysis consist of three test activities selected to represent several test levels and organizational units. The test activities are selected because of their frequency and costliness in comparison to other test activities at the same test level. Since the studied function belongs to one of the functional areas of Application software development and is not explicitly tested at Core software development, this organizational unit is not part of the sub-case. The sub-case is further limited to one project and two platform variants.

In summary, the case is a space, set up by the four dimensions:

- Part of product
- Organizational units (core software, application software and product composition)
- Test levels (feature test, integration test and system test), and
- Configurations (platform, product variant, release and product packages)

The feature testing is carried out by the functional area team, which consists of 17 people, all of whom are working with both development and test and have a common responsibility for the quality of the feature. A team of ten testers have the responsibility of integration testing. In the product composition organization, testing is distributed over many teams specialized on different markets.

In the case study, testing is studied under a period of 22 weeks (from bring up of the Android source to release of the feature to which it belongs). The test overlay is analyzed at five different levels of abstraction of the test cases, as explained in Section 3. All feature testing is included in the study, as well as the major part of the integration testing (the commonality testing before realization into 6 variants) and a minor part of the testing at product level.

2.5. Theoretical frame of references

In this work, we frame the research based on Pohl *et al*'s concepts of *commonality* and *variability*. SPL engineering offers a systematic approach for handling variability and for parallel development of several product variants derived from a common base. Systematic reuse in terms of testing could refer to reuse of test artifacts e.g. test models or test cases or to reuse of test results.

In our work, we extend the testing part of the Pohl model to include three dimensions for test variability, see Figure 1 [15]:

1. The traditional testing at different levels (unit, integration, system etc.)
2. Regression testing as the system evolves over time, and
3. Testing over the variation in the product space.

The variability across the three dimensions entail a high risk of costly redundancy in the testing and also the reverse: that important aspects of the differences between the tested artifacts are overlooked. However, in the studied case, the processes for commonality and variability are not distinctly separated from each other as in Pohl's conceptual model, which seems to be the case in most real-life product line processes.

2.6. Propositions

Propositions are predictions about the world that may be deduced logically from theory [17]. Below, we define propositions for the current study. P1 is derived from the theory in Section 2.5 and P2 is a basic assumption about redundant tests. P3–P8 cannot be derived from an explicit theory, but rather from our general assumptions underpinning the study. Advised by Verner *et al.* [20] we link propositions to research questions.

P1 The amount of overlaid testing is high in a product line context [RQ1]
P2 Redundant tests do not detect faults [RQ1]
P3 Distribution of test responsibilities causes overlaid testing [RQ2]
P4 Parallel development causes overlaid testing [RQ2]
P5 Insufficient delta analysis causes redundant testing [RQ2]
P6 Poor documentation and structure cause redundant testing [RQ2]
P7 Redundant testing can be avoided if test selection is supported by visualization of test data. [RQ3]
P8 The visualization must be correct, understandable and relevant, in order to fulfill its purpose [RQ3]

2.7. Concepts and measures

In this section, we define some concepts and terms which we use throughout the paper.

Definition 1. A *software product line* is a "set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way" [5]. An *emerging* software product line is one where the "prescribed way" is not well defined yet.

Definition 2. *Commonality* denotes the shared set of features while *variability* denotes the possible specializations.

Definition 3. A *coverage item (CovI)* is an entity describing the scope of a test or set of tests. All test cases are designed to cover at least one CovI. The scope is defined in terms of part of product (e.g. feature) and test purpose (e.g. response time). A coverage item may be defined hierarchically, i.e. a high-level coverage item consists of several low-level coverage items.

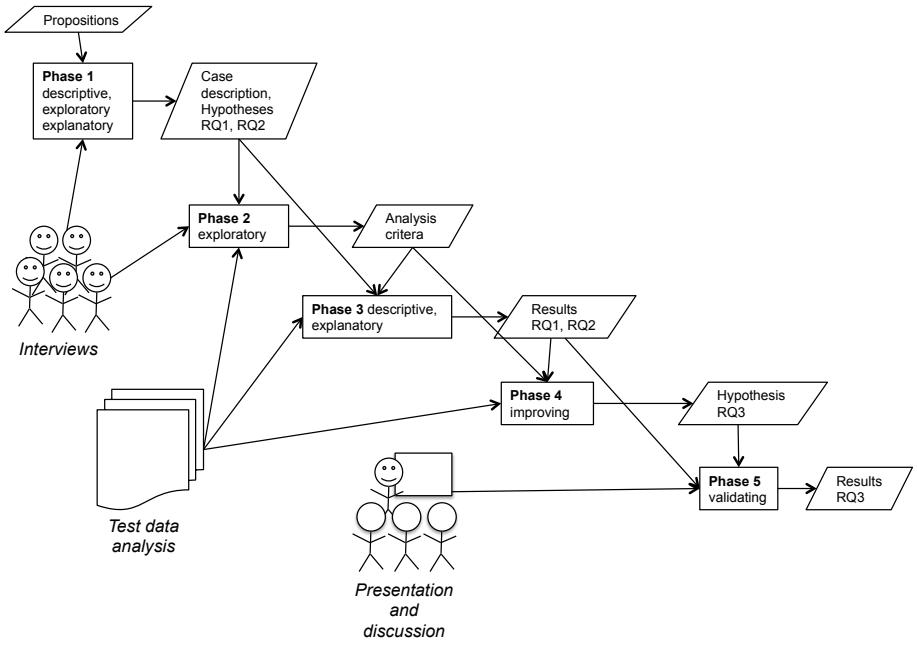

Figure 4: The different phases of the case study

Definition 4. A test case executes a specific version and variant of the software system or subsystem. The *delta* denotes the change between two versions or the difference between two variants.

Definition 5. *Test overlay* refers to a test case, which partly or fully covers another test case. Test *design* overlay refers to multiple test cases covering the same coverage item, although they may cover different coverage items at a less abstract level. Test *execution* overlay refers to multiple executions of test cases, covering the same coverage item.

Definition 6. *Redundant tests* refers to overlaid test cases where any differences between the tests do not affect the outcome of the tests. Test *design* redundancy refers to multiple test cases designed to cover the same coverage item at the lowest level of abstraction. Test *execution* redundancy refers to multiple test executions, where neither differences in coverage items nor delta between test objects affect the outcome of the test.

2.8. Data collection and analysis

The case study comprises both qualitative and quantitative observations, and data collection and analysis is carried out in an iterative manner, successively increasing our understanding about the case. Five different phases describe the data collection and analysis procedure from our starting point to the conclusions drawn, see Figure 4.

Table 1: Interviews held in the study, with different roles in the organization: Configuration manager = CM, Manager = M, System architect = SA, Test lead = TL, Test architect = TA.

Phase	Topic	Core SW org.	Application SW org.: *feature testing*	Application SW org.: *integration testing*	Product composition org.
1	Test strategies and activities	TA	TL	TA	TA
1	Variability space	TA	SA, TL	CM, TA	M, TL, TA
1	Risk for test redundancy	TA	TL	TL, TA	TA
1	Configuration management activities			CM	M, TA
1	Challenges in release management				M
1	Branching strategies			CM	M, TA
2	Overview of feature		SA, TL		
2	Test case design		TL		TL
2	Test selection		TL	TA	TL
2	Test documentation		TL	TA	TL

In the *first phase* the general context was described and explored for the purpose of framing the case study and increasing our understanding of the general case, product line testing at the case company. Interviews were held with nine persons from different units of the organization: one from the core software development organization, five from the application software development organization (of which two were from the feature team responsible for the sub-case function) and three from the product composition organization. The interviewees covered a variation of roles: three test architects, three test leaders, one software architect, one configuration manager, and one line manager, see Table 1. The interviews had the format of open discussions covering a number of high level topics listed in Table 1. During the interviews notes were taken and most interviews were recorded for later reference, but not transcribed. Process documentation and training material were also a source of information in the first phase.

A model describing the case context, was created and gradually improved for each interview, eventually leading to Figures 2 and 3. This was done based on notes from interviews, the documentation referred to in each interview, and the responses to the model as it was presented to the interviewees. In addition to the case description, the outcome of this phase was the scoping and final definition of the sub-case and hypotheses regarding the documented test executions in the sub-case. The hypotheses originate from our initial propositions as well as from the observations in the first phase.

In the *second phase* the sub-case was further explored. The test management database was manually searched for test cases and test executions related to

the selected case. The test documentation was analyzed with an exploratory approach:

1. Search for test cases, testing the function
2. Identify parameters of variability in test executions
3. Classify test cases with respect to those parameters
4. Define relevant dimensions of coverage items with respect to identified classes of test cases
5. Identify a relevant abstraction hierarchy of coverage items with respect to identified classes of test cases

In addition, three of the interviewees were asked additional questions about the sub-case.

The outcome of this step was the analysis criteria, used for testing the hypotheses from the previous phase. More details and the result of this step is given in Section 3.

In *phase three*, test overlay was assessed from different perspectives and at different abstraction levels (RQ1). This was done by applying the analysis criteria from phase two to the test execution data. The hypotheses together with this quantitative assessment of different types of test overlay (RQ2), see results in Section 4, formed the basis for the qualitative analysis in Section 5. This analysis was inspired by Miles and Huberman's graph models [12].

In the *fourth phase*, proposals for visualization of test coverage and overlay (RQ3) are derived. These proposals originate from the analysis criteria (how to capture the actual purposes of the tests?), the test documentation (which information is available?), and our conclusions regarding overlay and redundancy (RQ 1 and RQ2).

The proposals from phase four are partly validated in the *fifth phase* when presenting our results at three different occasions for different test architects and managers. The quantitative assessment were presented as well as our proposals for visualization. The responses and questions from the practitioners given at these occasions were also part of the input to the study.

2.9. Validity

This section discusses possible threats to the validity of the study, based on Runeson *et al.*'s guidelines [16], and reports actions taken to reduce the threats, where feasible.

Construct validity concerns the match or mismatch between the study context and the researcher's context and research questions. Of special interest to construct validity is the definitions of terms and concepts in the case context, vs. the research context. The authors of this paper have both spent considerable time in the case study context to adopt their terminology, and then transformed it into generally accepted terms in the research domain. Specific terms of interest include:

- *Coverage,* which we refer to as design and execution coverage, respectively, which is fairly well accepted in the case, although not perfectly.

- *Coverage item,* which here is a more general term than used in research, where it is often related to code coverage.

- *Redundancy,* which is used in a general sense in the case, while we here have a very precise definition of redundancy.

- *Software product line,* which in the software engineering literature is more of a technical perspective, while in this context it is very much a market approach.

We have continuously adjusted our understanding of the case terminology related to the research terminology in order to be as precise as possible. Further, the combined use of qualitative and quantitative data helps triangulating the observations and improve the construct validity.

Reliability concerns the extent to which the research is dependent on specific researchers. This is a threat in any study using qualitative (and quantitative!) data. The observations are of course filtered through the perception and knowledge profile of the researchers. Counteractions to these threats are that two researchers are involved in the study, and the observations are triangulated with quantitative and qualitative data. Another threat to the reliability is that the study design is very flexible, with several options in every step of the study. However, the overall research goal is kept the same during the course of the study, ensuring that the overall results of the study are reliable, although parts of the study would most probably have be done differently with another set of researchers. Furthermore, within the organization, they conducted a more pragmatic study in parallel on the topic, and the results from that study is well in line with this, strengthening the reliability of this study.

Internal validity is concerned with casual relationships among factors. In our case, the quantitative analysis is not interpreted in isolation, and it is not even feasible to infer statistical analysis, due to the incompleteness of the data. The analyses about casual relationships are instead based on qualitative analysis to generate hypotheses, which are validated using quantitative data. Feeding back the analysis results to interviewees is another action taken to reduce internal validity threats.

External validity regards the ability to generalize beyond the studied case. Each case of course has their own specifics, and in that sense there is no general case. However, some key characteristics of the case may be general and, for other cases with similar contexts, the results may be used as a reference. In order to allow external comparison, we have presented the context as clearly as possible, given the confidentiality constraints we have. On the risk side, there are so many variation factors in the context, that we may have focused on other than the key ones. Only replicated studies may help assessing the external validity of our study.

3. Analysis criteria

Based on our initial, exploratory study phase, the criteria for the analysis of the case are worked out. The analysis criteria include concept and terms which are assumed to be relevant for the deeper analysis of the case, as defined in Section 2.7.

3.1. Identification of test purposes

The basis for our analysis of overlay is the coverage items, i.e. the parts and aspects of the system which the testing is supposed to cover, and it is crucial for the relevance of our findings that our identification of coverage items are in line with the testers' purposes. We wanted to analyze test overlay at several levels of abstraction and from different perspectives. For the purpose of finding relevant definitions of coverage items, all test cases were classified with respect to their purpose and focus and mapped into a hierarchical structure. This was done in cooperation with the interviewees in the second phase of the study, see Table 1. Following is a list of variation factors of the executed test cases:

1. Focus – Which functionality is in focus.
2. Purpose of the test – Six different types of tests were found: duration, functionality, interoperability, performance, power consumption and stress tests.
3. Interaction – Except for plain tests of the case function or its variants, interaction with ten other important functions were tested.
4. Abstraction level – There is always a possibility to detail the testing further and add variations to a test case.
5. Software version – New versions of the software are released for testing, approximately twice a week.
6. Hardware – Testing on two different hardware types is covered by the case.

We decided to include the first four variation factors in our definition of the coverage item while the latter two are considered variations in the test object and as such possible subjects for delta analysis with respect to the coverage items.

3.2. Definition of coverage items

The four identified factors of variation can be described by two-dimensional coverage items. One dimension represents the *focus* of the test, such as a specific function, and the second represents the *purpose* of the test which could be, for example, testing the interaction with other features. The two parameters, focus and purpose, are in turn hierarchical values, see Figures 5 and 6, which enable a pairwise coverage analysis of general and specific test requirements at different levels of abstraction.

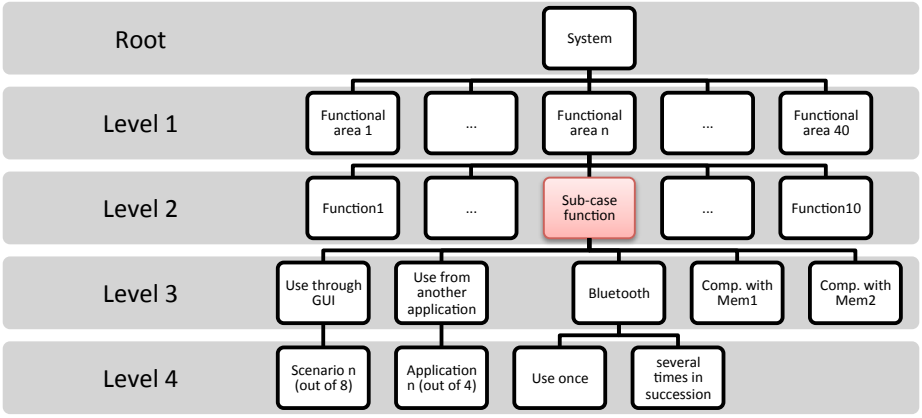

Figure 5: The hierarchical structure of the 'focus' parameter in a coverage item. The first level of details contains the divisions of the system in functional areas. Level 2 contains the main functions of a functional area. The 'sub-case function' node at level 2 is included in the analysis in this study.

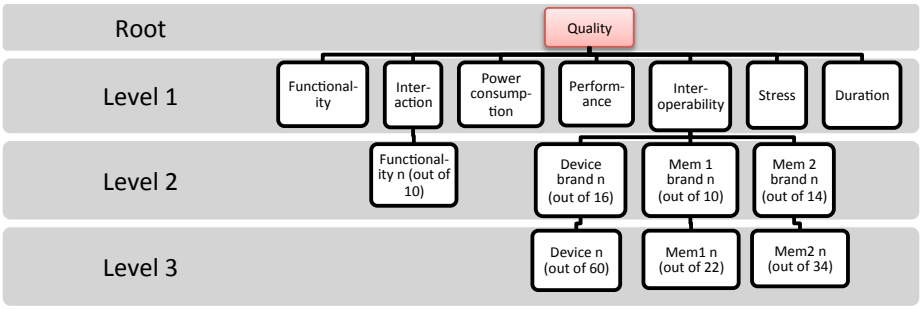

Figure 6: The hierarchical structure of the 'purpose' parameter in a coverage item. The first level of abstraction includes the main quality attributes. At level 2 the 'Interaction' node is extended with different interacting functionalities and the 'Interoperability' node is extended with different brands of the two memory types and of the communicating devices. These nodes are then extended further with details on different models. Here the root node represents the scope of the analysis.

3.3. Definition of data points for analysis

The 192 test cases in our sub-case cover coverage items distributed over five different levels of abstraction. This is illustrated in Table 2, where the columns represent the different levels of the 'purpose' parameter and the rows represent the levels of the 'focus' parameter. The numbers in the cells denote the number of test cases, explicitly designed to cover the corresponding pair of parameter values. The basis for our analysis (i.e. the data points) is the five different levels covered by test cases. Note that in order to retrieve the total number of test cases covering a pair of parameter values, the test cases designed for lower levels are included, as well as the test cases with no further details defined. Purpose level 1 and Focus level 3 (P1@F3) is thus covered by 192 test cases while P2@F4 is only covered by 2 test cases, see Table 2.

The coverage item definitions used for this analysis is not a general proposal

Table 2: Number of test cases for each coverage item, composed of a 'focus' and a 'purpose' parameter. The rows of the table represent the levels of the 'focus' parameter and the columns represent the levels of the 'purpose' parameter.

Level		Purpose level 1 (P1)			Purpose level 2 (P2)		Purpose level 3 (P3)	**Total**
	Name	Interaction	Interoperability	5 quality attributes	10 interacting functions	40 brands	112 models	
Focus 3	GUI	0	0	10	2	0	0	12
	Other application	0	0	1	0		0	1
	Bluetooth	0	0	4	0	0	59	63
	Mem1/Mem2	0	1	16	15	4	53	89
Focus 4	8 GUI scenarios	0	0	13	2	0	0	15
	4 other applications	0	0	8	0	0	0	8
	Single/Multiple	0	0	4	0	0	0	4
Total		0	1	56	19	4	112	192

for structuring tests but a means to capture the testing goals within this context.

4. Quantitative data

In total we found 517 test executions of 192 *different* test cases, which tested the case function, over a period of 22 weeks, see Figure 7. Given the narrow scope of the case function under study, this is a high number of test executions and it was far more than expected by the test managers. The failure rate is generally low: 15 of the 517 test executions failed, with a concentration to week 3 where 11 executions failed in the system testing. Feature testing and integration testing run only one failing execution each. The quantitative data is summarized in Table 3 and the data for pairwise overlay between activities is summarized in Table 4.

4.1. Overlay in test design

We found 192 *different* test cases, testing the case function. At the highest level of coverage abstraction (Purpose level 1 and Focus level 3 – P1@F3), these 192 test cases cover 19 unique coverage items, see Figure 8. Hence, if overlay in test design is analyzed at this level of abstraction, 90% of the test cases could be considered overlaid since they do not cover any unique coverage items. In Table 3 it can be seen that Feature testing covers 7 CovI:s with 18 test cases, integration testing covers 11 CovI:s with 33 test cases and system testing covers

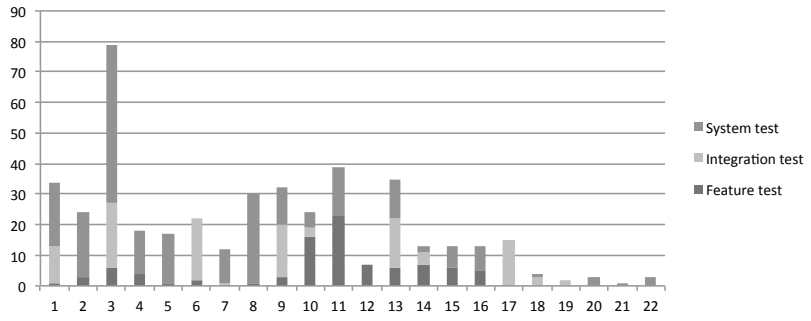

Figure 7: Test executions per test level and week.

15 coverage items with 141 test cases at this level. Furthermore the design overlay between test activities at this level is 40%.

A large share of the design overlay identified at the highest level of abstraction (P1@F3) can be attributed to the variability of the test cases, i.e. most of the test cases are different variants at a more detailed level of coverage analysis. There are, for example, 112 different system test cases designed at level P3@F3 to test the function in terms of compatibility with different models of devices and types and sizes of memories.

Decreasing the abstraction level of the 'purpose' parameter to P3@F3, there is no overlay between the two test levels: integration and system testing (see Table 3), and no overlay within feature testing (see Table 4). There is still design overlay within integration testing and system testing at this level, 33% and 63%, respectively (see Table 3).

4.2. Overlay in test execution

Overlay in test execution could origin both in overlay in the test design and the re-execution of a test case. However the overlaid test is not redundant if it has been affected by a change since the last execution. In this case study we did not have information about the delta between versions, and hence we could only measure an upper bound of redundancy.

At the highest level of abstraction (P1@F3), 517 test executions tested the case function. 96% of these are overlaid. The share of overlaid tests remains high even in the analysis at lower abstraction levels.

5. Existence and causes of test overlay – RQ1 and RQ2

This chapter reports the analysis related to the research questions on existence and causes of test overlay.

Table 3: Coverage data and overlay *within* each test level for the coverage items. Legend: F = Feature testing, I = Integration testing, S = System testing, Tot = Total, O = Overlay

Level		Purpose level 1 (P1)					Purpose level 2 (P2)					Purpose level 3 (P3)				
		F	**I**	**S**	**Tot**	**O**	**F**	**I**	**S**	**Tot**	**O**	**F**	**I**	**S**	**Tot**	**O**
Focus level 3 (F3)	#Executions	91	191	235	517		12	83	180	275		0	0	172	172	
	#Failed executions	1	1	13	15		0	1	13	14		0	0	13	13	
	Failure rate	1%	1%	6%	3%		0%	1%	7%	5%		0%	0%	8%	8%	
	#TC:s	18	33	141	192		3	15	117	135		0	0	112	112	
	#Covered CovI:s	7	11	15	19	14	3	10	43	54	2	0	0	112	112	0
	Coverage	20%	31%	43%	54%	74%	1%	4%	17%	22%	1%	0%	0%	20%	20%	0%
	Design overlay	61%	67%	89%	90%	8%	0%	33%	63%	60%		0%	0%	0%	0%	
	Execution overlay	92%	94%	94%	96%		75%	88%	76%	80%		0%	0%	35%	35%	
Focus level 4 (F4)	#Executions	31	85	18	134		0	12	0	12						
	#Failed executions	0	0	0	0		0	0	0	0						
	Failure rate	0%	0%	0%	0%		0%	0%	0%	0%						
	#TC:s	8	13	6	27		0	2	0	2						
	#Covered Items	8	12	5	20	5	0	2	0	2	0					
	Coverage	9%	13%	5%	22%	25%	0%	0%	0%	0%	0%					
	Design overlay	0%	8%	17%	26%	71%	0%	0%	0%	0%						
	Execution overlay	74%	86%	72%	85%		0%	83%	0%	83%						

Table 4: Pairwise test design overlay *between* test levels for each of the five coverage items.

Coverage items	**Overlay between pairs**			**Unique coverage**		
	F/I	**I/S**	**S/F**	**F**	**I**	**S**
P1@F3	13%	44%	29%	14%	18%	20%
P1@F4	5%	13%	18%	63%	75%	20%
P2@F3	8%	0%	5%	33%	100%	98%
P2@F4	0%	0%	NA	NA	100%	NA
P3@F3	NA	0%	0%	NA	NA	100%

5.1. Amount of test overlay

The context for our main case under study showed to be very variability-intensive, as reported in Section 2.3, and we expected a large amount of the testing to be overlaid (Proposition 1). This belief was also shared by most of the interviewees. All but one of the nine interviewees expressed a feeling that a lot of overlay testing was carried out and they helped us identify critical test activities for the in-depth analysis. This led us to state two hypotheses: H1) *Among the test executions a large share is overlaid and potentially redundant* and consequently: H2) *the failure rate of the testing carried out is low*, since redundant test cases are not expected to fail (Proposition 2).

Both hypotheses were to some extent confirmed by our quantitative data. Four out of five data points show more than 80% overlay, i.e. 80% of the test executions were re-executions of tests for coverage items covered previously in the project. The remaining data point shows a total overlay of 35%. The data

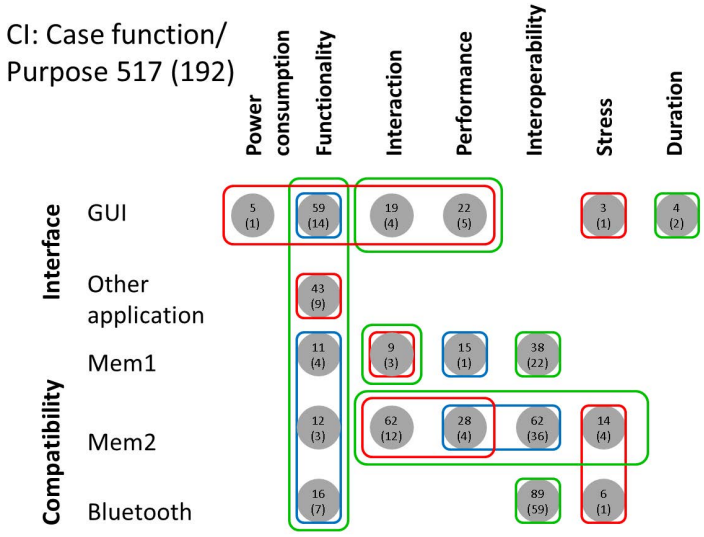

Figure 8: Coverage items covered by the feature testing (blue outline), integration testing (red outline) and product testing (green outline). Numbers within parentheses is the number of designed test cases and the numbers without is the executions.

point with less overlay represents the lowest level of compatibility tests with a lot of variant specific test cases. Note, however, that this is just an upper limit for the redundancy according to our criteria, defined in Section 3. No consideration has been taken to the changes between versions or the delta between product variants (two product variants were included in the analysis). The numbers indicate where improved change impact analysis or delta analysis could be beneficial. The failure rate is low as well: 1% at feature testing and integration testing and 6% at system testing. The concentration of failed executions to one test session and four variants on the memories at analysis level P2@F3 is also an indicator of redundancy.

5.2. Factors causing overlay

In the interviews, several factors increasing the test overlay, and consequently the risk for redundant testing, were pointed out:

1. Absence of complete requirements specifications – *"Requirements are not very detailed; main requirements are features from which user stories are developed by the feature team in cooperation with the scope owner."* (System architect – Feature testing)
2. Redundancy in requirements – *"Beside the user stories, detailed technical specifications (Req:s) exist. This combination introduces redundancy in requirements. Quality requirements are handled in another [organizational*

unit] which in turn put requirements on this [organizational unit]. Requirements specification is extended post hoc based on operators error reports." (System architect – Feature testing)

3. Legacy – The feature test suite was originally developed at another site. Due to frequent changes in the organization, responsibilities change.
4. System testing of generic areas – *"Product testing do a lot of duplicate testing in all generic areas since they are supposed to work with the customer requirements. Many of the customer requirements are the same."* (Test Architect – Core software) *"With Android a lot of the product verification gets double since the risk in customization does no longer exist.* (Test Leader – Integration testing)
5. The use of static test suites – Integration testing repeatedly runs the same test suite.
6. Parallel testing – *"They test the platform. We test it through our application and at the same time the whole is tested in the main branch."* (Test Leader – Feature testing) *"Unless the customer have specific customizations for an application, they can reuse test results from application software and from core software.* (Test Architect – Core software)

These factors are in line with our initial propositions and form a basis for our continued analysis together with the other qualitative data (process documentation and test artifacts). A chain of causes and effects which are relevant in our sub-case was outlined, see Figure 9, and lead to some additional hypotheses regarding our quantitative data: H3–H6 in Table 5.

The lack of complete requirements specifications in combination with a constantly changing organization is interpreted as "poor documentation and structure" (Proposition 6) and as such a cause of test design overlay in feature testing. The constant evolution of organization, processes and tools affects all three units of analysis and consequently they all have to deal with legacy, which in turn increases the inconsistency in test documentation which may also count as "poor documentation and structure". Integration testing and system testing is distributed between several teams of testers which further increases the risk of design overlay within those two activities (Proposition 3). According to the same proposition there is overlay between all three test activities.

Two types of delta analysis is referred to as insufficient in the interviews: the change impact analysis in integration testing and the commonality analysis in the system testing. Thus there is an increased risk for execution redundancy within these two activities (Proposition 5). Since a static test suite was used for regression testing in feature testing as well, this risk is present there too. The parallel work lead to execution overlay (Proposition 4) only if there is an overlay in design between the test activities.

In addition to the hypotheses in Table 5, interviewees expressed their assumptions. It was explicitly said in the interviews that there is a high degree of design overlay between feature testing and integration testing (in line with H4a) and that the problem of test overlay was solved at system test level thanks to a well defined process and good communication (in contrast with H4b and H4c).

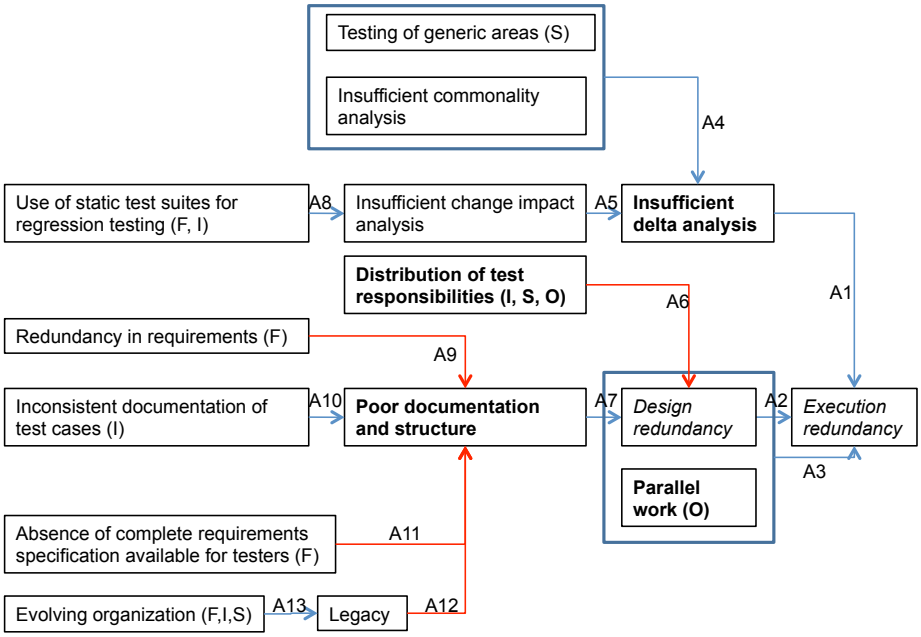

Figure 9: Graph illustrating the hypotheses derived from the qualitative analysis. Text in bold represents our initial propositions. Letters within parentheses denote which units of analysis the hypotheses concern. Arrows A1–13 are linked to the hypotheses in Table 5. Red arrows indicate relations partly contradicted by the quantitative data. F = Feature testing, I= Integration testing, S = System testing, O = Overlay.

5.3. Test of hypotheses

The analysis in this section is structured according to our initial propositions (i.e. nodes with bold text in the graph, see Figure 9). Each hypothesis is related to one or more chains of arrows within the graph and one arrow may relate to more than one hypothesis.

5.3.1. Distribution of test responsibilities

The proposition that distribution of test responsibilities causes overlaid testing (P3) is not unconditionally true. Even though a general overlay in test design is confirmed by the quantitative data, see Design overlay in Table 3, there is some variation between abstraction levels as well as between test levels. The overlay within a test level is more frequent than the overlay between test levels for two out of three data points, where design overlay is present. The total design overlay at level P1@F3 is 90% of which only 8% can be attributed to overlay between the test levels. On the other hand do these 8% represent 74% of the reached coverage. At level P1@F4 the relationships are the reverse: 71% of the design overlay can be attributed to overlay between test levels but only 25% of the reached coverage.

Table 5: List of generated hypotheses linked to the arrows, A1–13, in Figure 9.

H1	Among the test executions a large share is overlaid. (A1–3)
H2	The fault detection rate of the testing carried out is low. (A1–3)
H3	There is a high degree of design overlay within each of the test levels (A7, A12, A13): a) feature testing (A9, A11) b) integration testing (A6, A10) and c) system testing (A6).
H4	There is a high degree of design overlay between each pair of test levels: a) feature testing vs. integration testing b) integration testing vs system testing and c)system testing vs. feature testing.(A6)
H5	There is a high degree of execution overlay within each of the test levels: a) feature testing (A5, A8) b) integration testing (A5, A8) and c) system testing (A4).
H6	If H4 holds there is a high degree of execution overlay (A3)

To test hypothesis H4, the share of test overlay between the test levels is analyzed with respect to the reached coverage. Pairwise overlay is analyzed for three pairs of test levels (feature testing vs. integration testing, integration testing vs. system testing and system test vs. feature testing) and is related to the the aggregated coverage of the two overlaid levels analyzed. We also analyze the share of unique coverage in relation to the total coverage at each test level, see Table 4. Pairwise overlay occurs mainly at the highest abstraction levels i.e. the general test cases overlay between the different test levels while the detailed test cases varies in focus between the test level. Both integration testing and system testing have 100% unique coverage at their highest level of abstraction (detailed interaction tests at integration testing and compatibility tests at system testing). Feature testing does not reach to more than 63% of unique coverage at any level.

In our sub case, organizational distribution seems to have greater impact on overlay than geographical. Feature testing and integration testing both belong to the application software organization, while system testing belongs to the product composition organization. Both integration testing and system testing are distributed globally. At level P1@F3 the share of unique coverage is low for all three test levels (between 14% and 20%) i.e. most of the covered CovI:s at this abstraction level is covered at another test level as well, which supports H4 to some extent. H4b and H4c are supported, while H4a is not, which is in contrast with the interviewees' intuitions. The pairwise design overlay between feature testing and integration testing is less than 15% at all abstraction levels. The other two pairs has larger overlay at the higher abstraction level; between 29–44%.

The geographical distribution of system tests does not seem to prevent the different teams to design non-overlaying sets of test cases. Analysis at the lowest level of abstraction shows no overlay within system testing, contradicting H3c but 33% overlay within integration testing, see Table 3. This weak support of H3b is partly explained by the inconsistent document structure, see Section 5.3.2.

5.3.2. Documentation and structure

The proposition that poor documentation and structure causes redundancy (P6) is not unconditionally true. At the highest abstraction level of analysis (P1@F3) H3 is confirmed: There is 61% overlaid feature test cases; 67% overlaid integration test cases, and 89% overlaid system test cases, see Table 3. However if decreasing the level of abstraction the data does not fully support the hypothesis. No internally overlaid (i.e. overlay within a test level) feature tests exist on the more detailed levels at this level. This means that H3a is *not* level..level supported. Absence of complete requirements does not necessarily cause redundancy, neither does redundancy in the requirements. From the perspective of a higher abstraction level (the manager's or test leader's perspective) the testing may be unbalanced with respect to the coverage items but not much is redundant since there are small variations in most of the test cases. The relevance of this variation is however unclear.

The size of the test suites and the number of involved test managers determines the need for proper documentation. The highest degree of internal design overlay, also at the more detailed level of analysis, exists within the integration test suite. 33% of the integration test cases are overlaid even when the explicit variation in the test cases is considered at level P2@F3 which weakly supports H3b, i.e. inconsistent test documentation of test cases could cause test redundancy. However, legacy does not cause design overlay since design overlay is not observed in the feature testing. Only executed test cases are included in the scope of this study and among those there are no overlaid feature tests at the detailed levels. One difference between feature testing and integration testing may be the awareness of the inconsistency, which was expressed in the interviews in the case of feature testing. The integration test suite is larger and managed by testers from several teams, while the feature test suite is mainly managed by one person. Also the lack of complete requirements specification and redundancy in requirements seems to be manageable in the small.

5.3.3. Delta analysis

Visualization of test coverage, as exemplified in Figure 8, helps the testers and test managers overview the intended coverage and thus assess its feasibility. The high degree of overlaid test cases at level P1@F3 could be attributed to the many variants of test cases for compatibility tests, which also explains the low degree of overlay where this variation is considered in the analysis (P3@F3). Here the data confirms the lack of delta analysis of the customer requirements as a cause of overlay.

Hypotheses H5 and H6 regard overlaid test executions. The share of overlaid test executions is high at all test levels and at all granularity levels, independently of the share of design overlay. Hence we can *not* reject H5 and H6, stating that insufficient delta analysis causes redundant test executions.

5.3.4. Summary

In summary, geographical distribution of testing does not cause test overlay but organizational distribution might do. Poor requirements documentation

does not directly cause overlaid testing but poor test documentation might do. Insufficient delta analysis may cause a high degree of overaly within this context.

6. Visualization – RQ3

The third research question regards how to support test scoping decisions. It is assumed that redundant testing can be avoided with good visualization (proposition P7). The condition under which we assume this holds is that the visualization is correct (*correctness*), that the receivers of the information interpret the information correctly (*understandability*), and that the visualized information is relevant with respect to the decisions it is supposed to support (*relevance*) (proposition P8). Below, we elaborate on the visualization with respect to these three conditions. Our conclusions regarding visualization in this section are achieved with analytical reasoning based on the observed needs for improvement with respect to RQ1 and RQ2 as well as our experience in analyzing overlay within this context.

6.1. Correctness

In this study the documentation and visualization of test coverage, and consequently the overlay analysis, was based on the identified coverage items. It was clear from the interviews that in order for this visualization to be correct it is should cover not only the focus of the tests but also the purpose. *"We may use the same test case but with another purpose, such overlaps we cannot avoid"* – TA-core software.

With our two-dimensional and hierarchical structure of the coverage items, which was quite easy to visualize, we managed to capture all the documented variations regarding both focus and purpose in the 192 test cases. Hence it seems to be a useful model for documenting and visualizing test coverage. The model was the result of our exploratory analysis of the test set related to the sub case. There might of course exist non-documented intentions of test cases as well as different implementations of a test case depending on the tester and the situation. Such variation is impossible to visualize with a tool and could motivate a certain degree of visible overlay.

6.2. Relevance

In the study, several possible causes of redundancy were identified, see Section 5. These were related to insufficient delta analysis, distribution of test responsibilities, poor documentation and structure of tests, and parallel work, If any visualization could enable improvement of these situations, it is considered relevant.

Two types of *delta analyses* need improvement (Section 5.3.3): 1) change impact analysis between consecutively tested versions of the software, and 2) commonality analysis between the different possible variants of the software which are to be verified. In both cases the need is to support the de-scoping of tests by identifying overlay and possible redundancy. In the first case the task is to

reduce the amount of *test cases* while in the second case the task is to reduce the amount of *variants* on which the test cases should be executed.

Thus two different views could offer relevant support: one visualizing the priorities of coverage items and one visualizing the priorities of the variants. In both cases priorities could be calculated guided by existing regression testing techniques [9, 22]. However, most of these techniques are very context dependent, and in many cases only useful for small systems or small test cases [14].

Distribution of test responsibilities across organization raise high demands on communication between the parties if redundancy in testing shall be avoided. First of all, a common definition of coverage items is needed in order to clarify the division of responsibilities. The modeling of coverage items used in this study is one example of how to make these definitions transparent. Based on the definition of coverage items, a test design view visualizing the test scope of the different parties would offer support in pinpointing overlay. Some of the test overlay identified in our study could be attributed to such gaps in communication (Section 5.3.1), indicating a need for such support. In case of overlay in test design scope in combination with *parallel work*, a view of the aggregated test execution progress would support decisions on execution scoping over time.

Poor documentation and structure of tests prevents a proper visualization and could not be improved by visualization itself. On the other hand could transparent definitions of coverage items (i.e. using the CovI model) guide and improve the documentation. Exploratory testing is a part of the testing strategy in our case under study and the idea of strict test documentation was met with skepticism. However, the introduction of a clear structure and rules for documentation does not necessarily put requirements on the level of detail, and even in the case of exploratory testing some documentation is necessary.

6.3. Understandability

In order for a tester or manager to understand the coverage view, the amount and type of information must be adapted to their information needs. The two-dimensional coverage is simple to visualize with a matrix as we did in our analysis, see Figure 8, and the type and level of detail can be selected according to the tree views presented in Figures 5 and 6. In our case study this navigation between detail levels supported the qualitative analysis of the nature of existing overlay, as well as our communication with managers at different levels in the organization. It helped them identify test areas with unreasonably large amounts of tests.

6.4. Summary

In summary, the CovI model used in this study may be a useful model for documenting and visualizing test coverage. The model supports communication through increased transparency and enables relevant views to support test design and test planning decisions. Within this context the model was sufficient for covering the test variability in terms of focus and purpose of tests and it

enabled navigation between levels of details which in turn supports communication across organizational levels. Following views would be relevant to provide for the purpose of improving test scope selection within this context: test design coverage and test execution progress, priorities and dependencies between test cases as well as between different test objects.

7. Conclusions

An in-depth case study was launched for the purpose of investigating test overlay in a large-scale variability-intensive context. The testing context under study is complex in terms of the *large-scale* (millions of lines of code) and *variability* of the system (realized in hundreds of different system configurations), *distributed* and *parallel* development (both geographically and and organizationally), and *iterative* and *agile* development process. Testing is repeated in three dimensions: over *time* (regression testing) in *space* (variant testing) and over *testing levels*. Variability is realized at compile time and in runtime.

Conclusions drawn are based on our interpretation of both quantitative and qualitative observations. Following is a list of contributions of this paper:

The amount of overlay was in general large (RQ1) but varied with: different testing levels, different purposes of tests and the different abstraction levels of the coverage analysis. Overlay is measured in terms of repeated executions of tests with respect to coverage items (CovI:s). The CovI:s describe the focus and purpose of the test and enables coverage analysis at different levels of abstraction. The study shows a high degree of both design overlay and execution overlay at all levels of test.

Overlay seems to be caused by several factors such as (RQ2):

- Distribution of test responsibilities – Organizational distribution had greater impact than geographical.
- Inconsistent documentation of test cases – The importance of consistency in design and documentation of test cases seems to depend on the size of the test suite and the number of involved test managers. In contrast to the intuition of the practitioners, redundancy in requirements or the absence of a complete requirement specification did not cause design overlay in the testing.
- Insufficient delta analysis – Lack of commonality analysis of the variation in space as well as lack of regression analysis of the variation in time were the two main causes of overlaid test executions.

Visual decision support could be provided with (RQ3):

- Visualization of test design coverage – Both focus and purpose of test should be visualized.
- Visualization of priorities of coverage items as well as priorities of variants.
- Visualization of aggregated test execution progress.

Testing could be more effective by improving delta analyses and with a more consistent way of documenting the work, not saying testing has to be specified

in detail up front. Coverage items may be identified post hoc and work as a basis for test progress visualization which in turn could improve test selection decisions.

Acknowledgments

The authors are very thankful to the case company and its employees, for letting us access them and their data, as well as giving valuable feedback on our findings. We also thank Dr Mika Mäntylä at Lund University as well as the journal reviewers, for valuable review comments to an earlier version of the paper. The research was in part funded by the EASE Industrial Excellence Center on Embedded Applications Software Engineering, http://ease.cs.lth.se. Emelie Engström is a member of the SWELL research school, Swedish V&V Excellence, http://www.swell.se.

References

[1] ISO/IEC 9126-1:2001(E), International standard software engineering product quality part 1: Quality model. Technical report, ISO/IEC, 2001.

[2] Anneliese A. Andrews, Jeff Offutt, and Roger T. Alexander. Testing web applications by modeling with FSMs. *Software and Systems Modeling*, 4:326–345, 2005.

[3] Ulf Asklund, Lars Bendix, Henrik Christensen, and Boris Magnusson. The unified extensional versioning model. In *System Configuration Management*, volume 1675 of *Lecture Notes in Computer Science*, pages 793–793. Springer Berlin / Heidelberg, 1999.

[4] Gerardo Canfora and Massimiliano Di Penta. Service-oriented architectures testing: A survey. In Andrea De Lucia and Filomena Ferrucci, editors, *Software Engineering*, volume 5413 of *Lecture Notes in Computer Science*, pages 78–105. Springer Berlin / Heidelberg, 2009.

[5] Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, Boston, 2001.

[6] Emelie Engström and Per Runeson. A qualitative survey of regression testing practices. In M. Ali Babar, Matias Vierimaa, and Markku Oivo, editors, *Product-Focused Software Process Improvement*, volume 6156 of *Lecture Notes in Computer Science*, pages 3–16. Springer Berlin / Heidelberg, 2010.

[7] Emelie Engström and Per Runeson. Software product line testing – a systematic mapping study. *Information and Software Technology*, 53(1):2–13, 2011.

[8] Emelie Engström, Per Runeson, and Andreas Ljung. Improving regression testing transparency and efficiency with history based prioritization – an industrial case study. In *Proceedings of the 4th International Conference on Software Testing Verification and Validation*, pages 367–376. IEEE Computer Society, 2011.

[9] Emelie Engström, Per Runeson, and Mats Skoglund. A systematic review on regression test selection techniques. *Information and Software Technology*, 52(1):14–30, 2010.

[10] Emelie Engström, Per Runeson, and Greger Wikstrand. An empirical evaluation of regression testing based on fix-cache recommendations. In *Proceedings of the 3rd International Conference on Software Testing Verification and Validation*, pages 75–78. IEEE Computer Society, 2010.

[11] Mary Jean Harrold. Testing: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 61–72, New York, NY, USA, 2000. ACM.

[12] Matthew B. Miles and A. Michael Huberman. *Qualitative Data Analysis: An Expanded Source Book*. Sage, 1994.

[13] Kai Petersen and Claes Wohlin. Context in industrial software engineering research. In *Proceeding of the 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 401–404, 2009.

[14] Gregg Rothermel, Sebastian Elbaum, Alexey Malishevsky, Praveen Kallakuri, and Brian Davia. The impact of test suite granularity on the cost-effectiveness of regression testing. In *International Conference on Software Engineering*, pages 130–140, Los Alamitos, CA, USA, 2002. IEEE Computer Society.

[15] Per Runeson and Emelie Engström. Software product line testing – a 3D regression testing problem. In *Proceedings of 2nd International Workshop on Regression Testing*, 2012.

[16] Per Runeson, Martin Höst, Austen Rainer, and Björn Regnell. *Case Study Research in Software Engineering – Guidelines and Examples*. Wiley, 2012.

[17] Graeme Shanks. Guidelines for conducting positivist case study research in information systems. *Australasian Journal of Information Systems*, 10(1):76–85, 2002.

[18] Antti Tevanlinna, Juha Taina, and Raine Kauppinen. Product family testing: a survey. *SIGSOFT Softw. Eng. Notes*, 29(2):12, March 2004.

[19] Cheng Thao, Ethan V. Munson, and Tien Nhut Nguyen. Software configuration management for product derivation in software product families. In *Proceedings of the 15th IEEE International Conference on Engineering of Computer-Based Systems*, pages 265–274, 2008.

[20] June M. Verner, Jennifer Sampson, Vladimir Tosic, Nur Azzah Abu Bakar, and Barbara A. Kitchenham. Guidelines for industrially-based multiple case studies in software engineering. In *Third International Conference on Research Challenges in Information Science*, pages 313–324, Fez, Morocco, 2009.

[21] Eleaine J. Weyuker. Testing component-based software: a cautionary tale. *Software, IEEE*, 15(5):54–59, sep/oct 1998.

[22] Shin Yoo and Mark Harman. Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification and Reliability*, 22(2):65–143, 2012.

[23] Andy Zaidman, Bart Van Rompaey, Arie van Deursen, and Serge Demeyer. Studying the co-evolution of production and test code in open source and industrial developer test processes through repository mining. *Empirical Software Engineering*, 16(3):325–364, 2011.