

**Supporting Decisions
on Regression Test Scoping
in a Software Product Line Context
– from Evidence to Practice**



Emelie Engström

Doctoral Dissertation, 2013

Department of Computer Science
Lund University

Dissertation 42, 2013
LU-CS-DISS:2013-1
ISSN 1404-1219
ISBN 978-91-980754-1-0

Department of Computer Science
Lund University
Box 118
SE-221 00 Lund
Sweden

Email: Emelie.Engstrom@cs.lth.se
Typeset using L^AT_EX
Printed in Sweden by Tryckeriet i E-huset, Lund, 2013

© 2013 *Emelie Engström*

ABSTRACT

Large software organizations with a product line development approach face many challenges regarding testing. Test managers need to make tradeoffs along three dimensions of repeated testing (abstraction level, time and product space) and consider a range of goals and constraints. In order to keep pace with the decreased development time for new products, which is enabled by the systematic reuse, selective testing of product variants is necessary. A common industrial practise is to base test scoping decisions on practitioners' expertise and experience. However, as software systems grow in size and complexity, the combinatorial explosion of test possibilities makes it infeasible to assess the reasonableness of the decisions without tool support.

Research on regression test selection propose several systematic strategies for setting a proper test scope when verifying changes of previously tested code. The goal of our research is to improve test management and reduce the amount of redundant testing in the product line context by applying regression test selection strategies. However, despite extensive research on regression testing, gained knowledge has not impacted on industry practises. Thus a secondary goal of our research is to bridge the gap between research and practice in the field of regression testing. Test planning support, like any decision support system, must operate in a complex context and need to be pragmatic, adapted to the context and evolve incrementally within the context.

This thesis explores state of art and state of practice of regression testing and software product line testing, and proposes and evaluates a visualization strategy to support regression test scoping in the product line context. Two extensive systematic literature reviews are conducted as well as four empirical studies in close cooperation with industry. Through visualization of relevant information at a proper level of detail, test management in general may be supported. A visual analytics tool for test management would also provide a framework which enables research based and context specific regression testing improvements.

CONTENTS

Preface	ix
Popular Science Summary in Swedish	xi
Acknowledgements	xvii
Thesis Introduction	1
Introduction	3
1 Introduction	3
2 Background, concepts and definitions	8
3 Research methodology	13
4 Research contributions	20
5 Related work	33
6 Future research directions	37
7 Conclusion	39
References	40
Included Papers	55
I A Systematic Review on Regression Test Selection Techniques	57
1 Introduction	57
2 Research Method	59
3 Results	65
4 Discussion	85
5 Conclusions and future work	88
References	90

II	A Qualitative Survey of Regression Testing Practices	97
1	Introduction	97
2	Method description	99
3	Analysis of the results	105
4	Conclusions	110
	References	112
III	Indirect Effects in Evidential Assessment: A Case Study on Regression Test Technology Adoption	115
1	Introduction	116
2	The regression test automation procedure	117
3	The case study	121
4	Discussion	125
5	Conclusion	126
	References	128
IV	Software Product Line Testing – A Systematic Mapping Study	131
1	Introduction	131
2	Research method	132
3	Challenges in testing a software product line	136
4	Primary studies	137
5	Classification Schemes	139
6	Mapping	140
7	Discussion	150
8	Conclusions	151
	References	153
V	Test Overlay in an Emerging Software Product Line – An Industrial Case Study	161
1	Introduction	162
2	Case study design	163
3	Analysis criteria	175
4	Quantitative data	178
5	Existence and causes of test overlay – RQ1 and RQ2	181
6	Visualization – RQ3	187
7	Conclusions	189
	References	191
VI	Supporting Test Scoping with Visual Analytics	195
1	Introduction	195
2	Case study	196
3	Prototype visualizations	198
4	Aspects of visual analytics	198
5	Conclusion	206
	References	209

PREFACE

This thesis is divided in two parts. The first part introduces the context of the research, describes the research methodology, summarizes the findings and discusses future research directions. The second part includes the six research papers on which the conclusions in the first part are based.

Publications Included in the Thesis

- I **A Systematic Review on Regression Test Selection Techniques**
Emelie Engström, Per Runeson and Mats Skoglund
Journal of Information and Software Technology 52(1):14-30, 2010.
- II **A Qualitative Survey of Regression Testing Practices**
Emelie Engström and Per Runeson
Proceedings of 11th International Conference on Product Focused Software Development and Process Improvement (PROFES'10), June 2010.
- III **Indirect Effects in Evidential Assessment: A Case Study on Regression Test Technology Adoption**
Emelie Engström, Robert Feldt and Rickard Torkar
The 2nd International Workshop on Evidential Assessment of Software Technologies (EAST'12), Sept 2012.
- IV **Software Product Line Testing – A Systematic Mapping Study**
Emelie Engström and Per Runeson
Journal of Information and Software Technology 53(1):2-13, 2011.
- V **Test Overlay in an Emerging Software Product Line – An Industrial Case Study**
Emelie Engström and Per Runeson
Journal of Information and Software Technology 55(3):581-594, 2013
- VI **Supporting Test Scoping with Visual Analytics**
Emelie Engström, Mika Mäntylä, Per Runeson and Markus Borg
Technical report LU-CS-TR: 2013-252

Contribution Statement

I am the main author of all included papers and as such responsible for running the research, dividing the work between co-authors and conducting most of the writing. The design of the SLR (paper I) and the industrial survey (paper II) was made in cooperation with the second author of the two publications while the other studies (Papers III–VI) was mainly designed by me with input from my co-authors. Ideas of the different studies originate from many sources of which my co-authors have their share. For example the starting point of the case study on technology adoption (paper III) was ideas picked up in a course on test automation held by the co-authors of that paper. I have also carried out the main part of the data collection in all studies, however, with support from other persons in different situations, e.g. specialists in terms of practitioners at the case companies (papers V and VI) or the librarian for identification of primary studies in the SLR (paper I); or fellow researchers in their participation in focus group meetings (paper II and VI) or in sharing the work load in e.g. screening papers of the SLR (paper I) or extracting data for visualization (paper VI). Analyses of data in all studies are primarily my analyses, however, validated by the co-authors. In addition, I have authored or co-authored nine papers which are related to but not included in the thesis. These are presented in Section 5.1 in the introduction part of the thesis.

Emelie Engström
March 2013

POPULAR SCIENCE SUMMARY IN SWEDISH



Pictures are available at www.flickr.com under Creative Commons Licences

Med karta och kompass genom testdjungeln

Av **Emelie Engström**

Institutionen för datavetenskap
Lunds universitet

De flesta av oss tar fungerande mjukvara för givet. Vi förväntar oss att banken hanterar våra insättningar och uttag korrekt, att bilens bromsar fungerar, att våra telefonsamtal kopplas rätt och att signalsystemet för tågtrafiken inte föranleder några olyckor.

En mobiltelefon är ett exempel på en liten, vardaglig pryl som innehåller ett mycket stort och komplext mjukvarusystem. En vanlig mobiltelefon in-

nehåller ungefär 10 miljoner rader programkod som utvecklats och testats av flera hundra ingenjörer.

Programkoden beskriver telefonens olika funktioner som till exempel hur

vanliga röstsamtal kopplas upp och genomförs, hur SMS skickas och tas emot eller hur kontakter hanteras i kontaktboken.

För att kunna veta att ett mjukvarusystem uppfyller våra krav och förväntningar måste tester genomföras. Tester där den faktiska användningen av telefonen simuleras. Varje funktion behöver testas enskilt och i samverkan med andra funktioner. Jag vill till exempel att samtal kopplas fram även när jag använder telefonen för att spela musik.

Testning en dyr flaskhals

Traditionellt räknar man med att uppemot 50 procent av kostnaderna för att utveckla mjukvara är kostnader för testning. Denna andel har ökat kraftigt på senare år på grund av att vi blivit bättre på att återvinna mjukvara på ett strukturerat sätt. Vi utvecklar produkter i familjer av liknande produkter. Dock har vi inte lärt oss återanvända testerna i samma utsträckning.

Målet för vår forskning är att hitta strategier för att identifiera vilka tester som är viktigast och vilka som är mer eller mindre meningslösa att genomföra. Särskilt fokus läggs på testning av produktfamiljer där mjukvaran återanvänds i mer än en produkt.

I en organisation som utvecklar stora mjukvarusystem upprepas liknande tester om och om igen. Först testas varje enskild del, därefter behöver delarna testas tillsammans efterhand som man bygger ihop systemet. Ofta utvecklas de enskilda delarna vidare efter att de infogats i systemet och proceduren måste upprepas. Om man dessutom vill utveckla flera varianter av

samma system behöver alla dessa också testas på samma sätt.

Oändliga alternativ

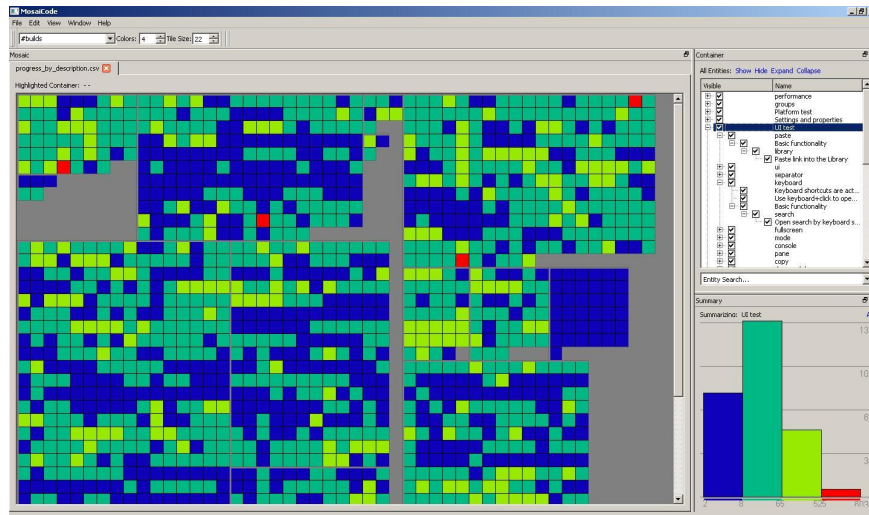
Inte ens ett litet system går att testa fullständigt. Tänk dig till exempel att du vill testa en enkel miniräknare och börjar så smått med $0+1$, $1+1$ till $999\ 999+1$ till $999\ 999+2$ osv till $999\ 999+999\ 999$. Denna systematiska genomgång av olika sätt att addera två heltal skulle kräva en biljon testfall vilket motsvarar 30 000 år av testning, om varje test tar 1 sekund att genomföra.

Många tester är inte heller nödvändiga att genomföra. Det kan vara rimligt att anta det räcker att göra några stickprov för att säkerställa att addition av två heltal fungerar. Kanske finns det kritiska övergångar (när antalet siffror i svaret inte längre får plats i displayen) som behöver en extra kontroll. Alla sådana antaganden baseras på erfarenheter av vanliga fel i mjukvara. De bygger också på kunskap om det aktuella systemet och skillnader gentemot liknande system som vi vet fungerar.

Problem uppstår när antalet variabler och alternativ blir för stort att hantera och det inte längre går att bedöma rimligheten i urvalen. Effekterna blir då antingen att mycket onödigt testning genomförs, för att vara på den säkra sidan, eller att testningen blir bristfällig. I många fall gäller båda.

Visualisering för bättre fokus

Det är dessa urval som är i fokus i avhandlingen. Målet är att kunna er-



Testtäckning av ett mjukvarusystem kan till exempel visualiseras med hjälp av ett rutnät där varje ruta motsvarar något som behöver testas och där färgerna talar om hur väl det är testat.

bjuda ett verktyg för att visualisera hur väl den testning som genomförs täcker kritiska delar av systemet. Testerna ska täcka systemet utifrån flera olika perspektiv: Har varje kodrad testats? Har alla funktioner testats? Tål systemet hög belastning? är det tillförlitligt?

Avhandlingen behandlar också införandet av ny teknik i en testprocess. Det kan finnas många faktorer som gör att det är svårt att införa nya teststrategier i en organisation. Testningen i sig är inte en isolerad företeelse utan hänger ihop med en mängd andra faktorer som t.ex. hur kraven ställs och dokumenteras, hur systemet designas och hur utvecklarna skriver programkoden. En förändring i den ena processen får konsekvenser även för relaterade processer. Därför måste förändringar införas i små steg och både direkta och indirekta effekter utvärderas. Vad händer till exempel med expertisen om man ersätter den med verktyg? Finns det risk

att verktyget används på fel sätt eller att man litar för mycket på verktyget?

Flexibla verktyg för varierande behov

Vi utvärderade våra visualiseringsstrategier genom att skapa prototyper och låta testledare utvärdera dem i samlingsgrupper. Alla som deltog i studien uppskattade det stöd verktyget gav för att planera testning och för att kommunicera beslut till andra personer i projektet. Man poängterade också vikten av att kunna interagera med verktyget.

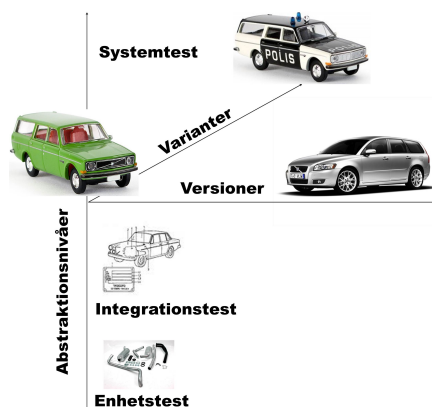
Det är viktigt med flexibilitet och att testningen kan analyseras från flera olika perspektiv. Behovet av information varierar mellan olika abstraktionsnivåer i ett projekt. En projektledare behöver en övergripande vy som omfattar alla perspektiv med få detaljer medan en testledare på enhetsnivå be-

höver en mer specialiserad och detaljerad vy. Utmaningen ligger i att lyfta fram rätt information i olika vyer och att identifiera hur testresultat från tidigare tester kan återanvändas.

I grunden handlar det om att spara pengar och öka säkerheten. Systemen kommer sannolikt att bli alltmer komplexa vilket kan leda till ohanterliga

kostnader för testning alternativt lägre kvalitet på våra mjukvarusystem. Med ett verktyg som samlar in och analyserar relevanta delar av den enorma mängd information som finns tillgänglig får ingenjörerna hjälp att fästa uppmärksamheten på de verkligt kritiska områdena.

Dimensioner av test



Abstraktionsnivåer: Utvecklingen av mjukvara inom en organisation distribueras ofta mellan olika team av utvecklare som ansvarar för en grupp av liknande funktioner (till exempel all hantering av kommunikation via bluetooth). Dessa minsta beståndsdelar testas enskilt med enhetstester. Därefter integrationstestas de tillsammans med andra komponenter av systemet för att se hur de samverkar och slutligen testas systemet i sin helhet mot de övergripande kraven på produkten med en fullständig systemtest. Antalet abstraktionsnivåer beror dels på storleken på systemen och dels på på hur man väljer att organisera utvecklingen.

Tid: Mjukvaran uppdateras kontinuerligt och nya versioner av både komponenter och delsystem integreras in i det aktuella systemet. Förändringar i en del av systemet kan påverka oförändrade delar och sådant som redan testats behöver testas om igen.

Rum: När mjukvaran återanvänds i flera liknande produkter återfinns dessutom flera varianter av både komponenter, delsystem och slutgiltiga produkter. Valmöjligheter kan finnas på alla nivåer och valen kan göras både under utvecklingen (till exempel anpassningar till olika hårdvarukomponenter) och efteråt av användarna genom personliga inställningar.

ACKNOWLEDGEMENTS

The work presented in this thesis was partly funded by the Swedish Governmental Agency for Innovation Systems under grant 2005-02483 for the UPPREPA project, and partly funded by the EASE Industrial Excellence Center on Embedded Applications Software Engineering, <http://ease.cs.lth.se>.

I am grateful to all people who, in different ways, contributed to the completion of this thesis. First of all, I would like to thank my supervisor *Prof. Per Runeson* for excellent guidance. I appreciate your generosity and positive attitude. Thanks also go to *Prof. Björn Regnell* and *Dr. Carina Andersson* for their co-supervision.

All studies reported in this thesis have been conducted in cooperation with other researchers to whom I also want to express my gratitude. *Dr. Mats Skoglund*, *Dr. Robert Feldt*, *Dr. Richard Torkar*, *Dr. Mika Mäntylä* and *Markus Borg*, it has been a pleasure! I'm also grateful for all effort spent reviewing manuscripts of the study reports as well as the thesis introduction. In addition to the many anonymous reviewers, I appreciate the valuable comments from *Prof. Per Runeson*, *Prof. Björn Regnell*, *Dr. Mika Mäntylä* and *Prof. Sebastian Elbaum*.

Many industry practitioners have been involved in this research. I would like to thank the case companies for letting us access their data, and all who have participated in interviews, focus group meetings and questionnaires, *SPIN-syd* (software process improvement network in Southern Sweden) and the *SAST* network (Swedish Association for Software Testing). I would like to address a special thanks to *Dr. Magnus C Ohlsson*, *Dr. Greger Wikstrand*, *Fredrik Scheja*, *Johan Hoberg*, *Per Beremark* and *Charlotta Nordmark* for sharing your experiences.

During the project, I have had the privilege to meet and get inspired by many fellow researchers. I would like to express my gratitude to the participants in the EASE Theme D project and in the SWELL (Swedish V&V Excellence) research school. I am grateful to *Prof. Laurie Williams* and the *Software Engineering Realsearch* research group at *NCSU* for the opportunity to visit your group last spring. Finally, A big thanks go to my colleagues at the department of computer science and especially the *SERG* (Software Engineering Research Group) for the inspiring and developing work environment.

Finally, I'm so grateful for my family and friends who supports and encourages me irrespective of what I accomplish. It's a true blessing to have you all in my life. *David, Miriam* and *Anton*, you challenge my thoughts more than anyone else but most of all you teach me what is important in life. Last but not least I want to express my deepest gratitude to *Jonas* for always standing by my side.

In God we live and move and have our being - To Him be the glory!

Emelie Engström
March 2013

THESIS INTRODUCTION

INTRODUCTION

1 Introduction

1.1 Software testing of large, complex and evolving systems

Since the 1960s software has changed from being small (100 lines of code) isolated control sequences handled by a few experts, to becoming large, complex, interacting systems handled by non-experts. One example: Android, on which many mobile phones are built, comprises more than 10 million lines of code. Similarly, development strategies are continuously adapted to meet market demands and manage software complexity. Iterative development [74] and software product line engineering (SPLE) [14] are two examples of developments strategies aimed to meet the demands for frequent and customized releases of software systems. Both of them have major impact on test management since they force much repetitive testing of similar software. Iterative development leads to repeated testing in time. SPLE adds complexity to the case by adding the dimension of variability and thus repeated testing in space. In this thesis, SPLE refers to the systematic reuse of software to enable efficient development of several variants or customizations of software systems.

Software testing, i.e. dynamic execution of the software, is an essential activity to ensure quality of a software system and to support its development by detecting faults. However, even for a small system, extensive testing is infeasible and a selected test scope is set to guide each test run. A common industrial practice is to base the testscoping decisions on practitioners' expertise and experience. As software systems grow in size and complexity the combinatorial explosion of test possibilities makes it infeasible to assess the reasonableness of these decisions without tool support. Research on test selection proposes several systematic strategies for setting a proper test scope depending on the test situation [19, 44]. Specifically, regression test selection strategies aim at verifying changes of previously tested code (Paper I). Similarly regression testing strategies could verify the effects of the differences between variants or customization of the software. Thus, those strategies are in focus in this thesis in terms of solution proposals for the soft-

ware product line (SPL) testing challenge. Since the testing challenge in a SPLE context originates in several context factors, we do not isolate the problem nor the solution to an idealistic SPLE case but rather search for a pragmatic treatment of testing working in a complex, variability intensive and frequently changing software development context.

The main goal of this thesis is to provide understanding of the preconditions for SPL test scoping and to propose strategies to improve test scoping within such context. As a means to reach that goal the area of regression testing is studied in this thesis and three sub goals are defined.

- Bridge the gap between state of art and state of practice of regression testing.
- Understand the challenge of SPL testing.
- Find means to support regression test scoping in the SPL context.

Regression testing was proposed by Tevanlinna [125] to be applied not only on versions but also on variants in order to reduce the amount of redundant testing across products derived from a common software base. However, despite extensive research on regression testing (Paper I), gained knowledge have not had effect on industry practices (Paper II). It is essential to understand the reasons for this gap between research and practice in order to find relevant applications of regression testing.

The main part of this thesis is exploratory and empirical studies are conducted in collaboration with different industrial partners. The complexity of software development in general and of SPLE in particular motivates the research approach as well as the general lack of such studies within the field of software engineering.

1.2 Research questions and main contributions

A first step towards bridging the gap between state of art and state of practice of regression testing is to explore it. Thus the first two questions, RQ1 and RQ2, posted in this thesis focus on exploring the research on regression testing as well as the industry practice of regression testing. RQ3, focus on providing prescriptions in line with the first subgoal. Focus is then shifted to explain regression testing in the SPL context, RQ4 and RQ5. Finally, RQ6 focus on providing prescriptions regarding the main goal of the research presented in this thesis. The following six main questions are investigated:

- *RQ1 What do we know about regression test selection?* Regression testing is a relatively well researched area within software engineering. Many techniques have been proposed and evaluated empirically. Limited sets of regression test selection techniques have been compared in previous reviews of regression test selection [12, 61], showing inconclusive results. No single solution to regression test selection fits into all situations nor could any

single study evaluate every aspect of the effects of applying them. However, due to the significant amount of empirical studies on the topic a meta analysis could serve to abstract and generalize knowledge. To get a more complete view of what we empirically know about regression test selection techniques we launched a systematic review (Paper I).

- *RQ2 How is regression testing applied in industry?* Despite extensive regression testing research, systematic approaches are rarely applied in industry. Only few empirical evaluations of regression test selection techniques are carried out in an industrial context [36, 38, 122, 130]. Instead practices are based on experience. To retrieve a better understanding of real world needs and practices we conducted a qualitative survey (Paper II).
- *RQ3 How to bridge the gap between state of art and state of practice of regression testing?* results from the studies reported in Papers I and II highlights several challenges in applying evidence based regression testing in industry. Challenges originate in the complexity of the industrial context and relate to the generalization and communication of findings. These challenges are summarized and analyzed in Paper III which also reports a case study on the transfer of empirical evidence into practice (Papers I-III).
- *RQ4 What do we know about SPLT?* Research on SPLE started in the 90ies and has gained a growing interest the last 10 years. Early literature on SPLs did not spend much attention to testing [14, p278-279], but the research issue is brought up after that, and much research effort is spent on a variety of topics related to product line testing. Main challenges are the large number of tests, the balance between effort for reusable components and concrete products and the handling of variability. To get an overview of the challenges and current solution proposals we launched a systematic mapping study (Paper IV).
- *RQ5 What is regression testing in a SPL context?* Results from our industrial survey (Paper II) highlight the variation in regression testing situations. Regression testing is applied differently in different organizations, at different levels of abstractions and at different stages of the development process. The common denominator is the handling of repetitive tests of similar software. In a SPL testing is repeated over different variants in addition to the versions. To understand the regression testing challenge in the SPL context we studied test overlay in an emerging SPL through an in-depth case study (Paper V).
- *RQ6 How to support regression test scoping in a SPL context?* Our two exploratory studies on regression testing and software product line testing, respectively (Papers II and IV), together with the two literature surveys on

the same topics (Papers I and IV) picture the SP -testing challenge as a multi-dimensional regression testing problem [117]. With existing regression testing approaches it is possible to provide automated decision support in a few specific cases while test management in general may be supported through visualization of test execution coverage, the testing space and similarities and differences between previous test executions in different dimensions. This visualization approach is described in Paper V and evaluated in three different industrial contexts. The evaluation is reported in Paper VI.

The main contributions of the thesis are:

- *Increased accessibility to regression testing research, through a systematic literature review, for practitioners and researchers who wants to apply regression testing techniques into a specific context (Paper I)* – Twenty eight different techniques were compared and classified with respect to context applicability, method aspects and effects.
- *Increased understanding of the industrial context and needs, through a survey, with respect to regression testing (Paper II)* – In most development organizations regression testing is applied continuously and at several levels with varying goals. Challenges relate to test case selection, trade-off between automated and manual testing and design for testability.
- *A demonstration of how to transfer empirical evidence on regression testing into practice, in terms of a case study (Paper III)* – Advice for practitioners are summarized regarding the identification of preconditions, identification of relevant research and evaluation of effects. Research challenges involve accounting for indirect effects of adopting a new technology as well as matching the communication of empirical evidence with a relevant context description.
- *An overview of the state of art in software product line testing, through a systematic mapping study (Paper IV)* – Sixty four publications on software product line testing were analyzed and classified with respect to research focus, type of contribution and type of research.
- *Increased understanding of the industrial prerequisites for regression testing in a software product line context, through a case study (Paper V)* – To avoid redundancy in such complex and variability intensive context tool support is needed. Factors causing a high degree of test overlay are: distribution of test responsibilities, poor documentation and structure of test cases, parallel work and insufficient delta analysis.
- *A visualization strategy to support regression test scoping in a software product line context, based on a case study and evaluated in focus groups*

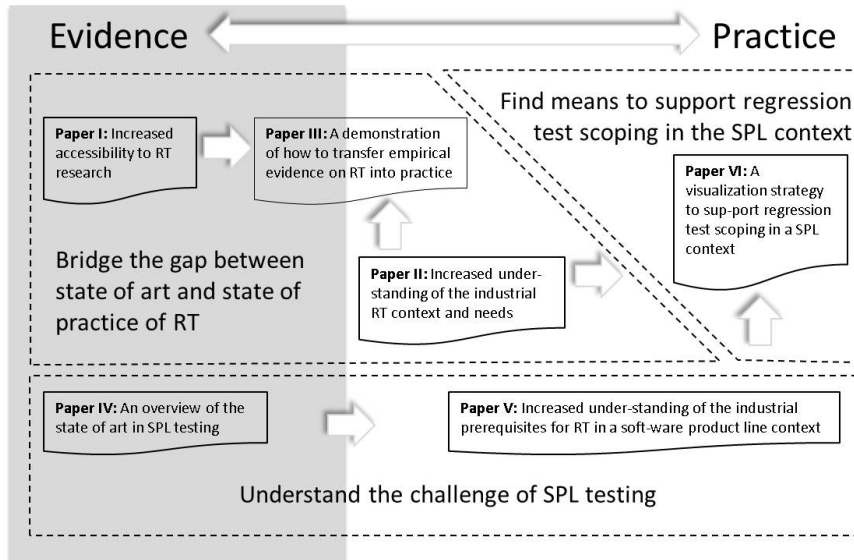


Figure 1: Overview of the contributions of the thesis.

(Paper V and VI) – Test execution history data were extracted and transformed to test coverage items (TCI:s), represented as sets of dimension and attribute values. TCI:s were further mapped to tiles of a mosaic window where the spatial position encodes the dimension variables and colors encode attribute values. All study participants confirmed potential usefulness of the visual analytics in support of test scoping as well as for communicating decisions with managers and subordinate testers.

The relationships between the various contributions are illustrated in Figure 1.

1.3 Outline of thesis

This thesis is composed of six papers and divided into two parts. The first part summarizes the contributions of the six papers, and the second part contains the papers in their entirety.

The thesis introduction is organized as follows. Chapter 2 describes the context of the research and Chapter 3 describes the research methodology. A summary of the findings are presented in Chapter 4 and of related work in Chapter 5. Chapter 6 discusses future research directions and Chapter 7 concludes the work.

In the paper section, six papers are included. The first three papers research the solution space, regression testing, from different perspectives. Paper I reviews the literature on regression testing, Paper II surveys industrial practices of regression

testing and Paper III studies regression test technology adoption. Paper IV and V research the problem domain, SPL testing. Paper IV reviews the literature and Paper V studies the industrial context. Paper V elaborates on how visualization may support regression test decisions in the SPL context. The implementation and evaluation of these ideas are reported in Paper 6.

Papers I, IV and V are published in the Journal of Information and Software Technology (IST) and Papers II and III are published and presented at the international conference on Product Focused Software Development and Process Improvement (PROFES) and the international workshop on Evidential Assessment of Software Technologies (EAST), respectively. Paper VI is submitted to a practitioner oriented journal.

2 Background, concepts and definitions

2.1 Software product line engineering

In SPLE mass customization is achieved through systematic reuse of artifacts throughout the development process. A software product line is defined as follows by Clements and Northrop:

“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.”

[22]

Commonality denotes the shared set of features while *variability* denotes the possible specializations, the *variants*. Pohl et al [99] further distinguish between the *domain engineering process*, in which the commonality and variability of the product line are defined and realized, and the *application engineering process*, in which the product variants, the *applications*, of the product line are defined and realized, see Figure 2.

The product line engineering concept is not unique for software engineering. Several examples of product lines are reported earlier in other engineering domains such as: Volkswagen cars [131], Xerox copiers [96], Swiss army knives and Swatch watches [128]. SPLE attempts to transfer the benefits of product line engineering, which enables both mass production and customized products, to the software engineering context.

However, in contrast to other engineering domains, the cost focus for SPLE is on the development process rather than the production. Hence, trade-offs relate to the increased effort in developing more generic artifacts versus the decreased effort in deriving new variants. Research on SPLE started in the 1990ies, and has

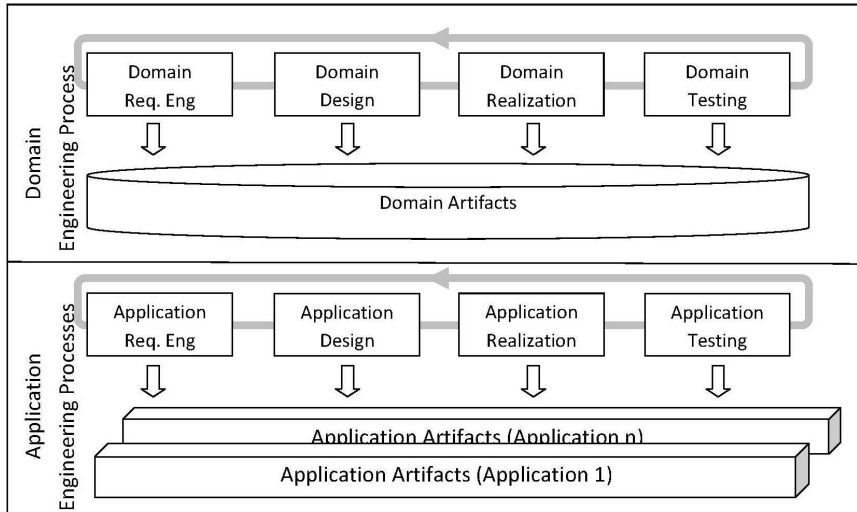


Figure 2: Domain and application engineering processes for software product line products, according to Pohl et al. [76], reprinted with permissions from Springer-Verlag.

attained growing interest over the last 10 years. Several benefits of applying SPLE have been reported, such as reduced lead times, costs and maintenance as well as increased flexibility [49,79].

Testing is still a bottleneck in SPLE. A common situation is product developers spending much time in testing each product variant as reported by Kato and Yamaguchi [63]. A few good examples are published on how the testing may be efficiently conducted [59], (Paper IV), but mostly, experience reports focus on other aspects than the testing [79].

2.2 Software testing

Basic testing concepts

Testing software involve several test activities: *test planning*, *test case design*, *test execution and analysis of test results* which are carried out differently at different stages of the development process (e.g. unit test, integration test and system test), see Figure 3. *Test planning* involves the specification of goals and choice of strategies for the testing. *Test case design* aims at specifying a set of test cases based on available information about the system, preferably in accordance with the previously defined goals and strategies. The specified set of test cases constitutes together with the software under test (SUT) the input to the *test execution*. Results from the test execution are analyzed to assess quality, identify faults and plan fur-

ther testing. In addition to the primary test activities a range of support activities may take place as well (e.g. test suite maintenance or test tool development).

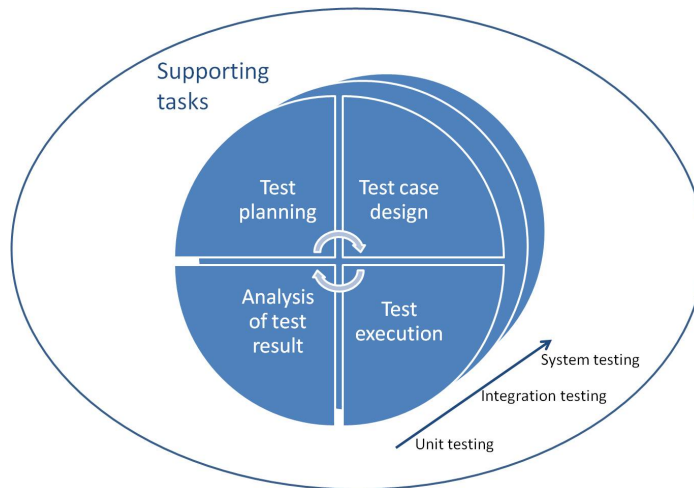


Figure 3: The different test activities. Primary activities are the planning of test, design of test cases, execution of test cases and analysis of results. these activities are carried out differently at different stages of the testing i.e. the levels of test (e.g. unit, integration or system).

Regression testing

Regression testing is conducted in order to verify that changes to a system has not negatively impacted on previously functioning software. The purpose is to detect *regression*, or decay of the system. IEEE standards define regression testing as:

“Selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements.”

[55,56]

A common regression testing approach is to reuse previously executed test cases and the main focus of regression testing research is how to maintain, select and prioritize among the existing test cases. In their survey on regression test techniques, Yoo and Harman [137] summarize definitions of *regression test case selection, prioritization and minimization* which were originally introduced by Rothermel and Harrold [47, 111, 114, 115] as follow:

Regression test selection, aims at identifying a subset of test cases that is sufficient to verify the effects of a given set of changes.

Regression test prioritization techniques rank test cases in their predicted order of contribution to the fault detection rate based on some criteria e.g. risk, coverage or test execution history.

Test suite minimization, or *reduction*, optimizes a test suite with respect to redundancy.

[137]

Out of these three groups the selection techniques are most researched [137]. The first paper on regression testing was published 1977 by Fischer [40] and until the beginning of 1990's regression testing research focused exclusively on the development of selection techniques. The concept of minimization was introduced 1993 by Harrold et al. [47] and prioritization techniques 1999 by Rothermel et al. [115]. The interest in prioritization techniques have been growing since, and since 2008 more papers have been published on regression test prioritization than on regression test selection [137].

An interest in evaluations and comparisons of regression testing techniques began with a publication on cost models by Leung and White 1991 [76] and an increasing trend of empirical evaluations in the form of evaluative case studies and comparative experiments started 1997 through Baradhi and Mansour [5], Rothermel and Harrold [112] and Wong et al. [134]. The empirical evidence base on regression test selection is systematically reviewed in this thesis (Paper I).

However, there is a gap between research and practice of regression testing. Only few empirical evaluations of regression testing techniques are carried out in a real industrial context. Techniques are context dependent and evaluations lack support for generalization between contexts. Moreover, while researchers mostly focus on selection, prioritization and minimization, there are other important issues too. Rooksby et al. [110] argue for the need for investigation and characterization of real world work, based on experiences from testing in four real projects. They conclude that improvements of current testing practices are meaningful in its specific local context and "cannot be brought about purely through technically driven innovation". Real-world needs and practices are investigated in this thesis (Paper II) through a qualitative survey as well as the application of empirical evidence in practice (Paper III) in a case study on the procedure of improving regression testing based on empirical evidence.

Regression testing is a frequent and costly activity in industry. Still industry practice on regression testing is mostly based on experience alone and not on systematic approaches. Onoma et al. [92] observe that regression testing is used extensively in industry, companies develop their own regression testing tools to automate the process and in some companies it is *not* a critical issue to minimize

the set of test cases to rerun. Some types of tests (e.g. customers' acceptance tests or safety standard fulfilling test) are often not subject to minimizations. Current trends in software engineering, such as software reuse over a family of products, constantly evolving software due to market and technology changes and specification free development, increase the need for and put specific requirements on regression testing. Today most testing could be regarded as regression testing. Berner et al. [8] observed that almost all test cases in a project were executed at least five times and 25% of the test cases were executed far more than 20 times in a project.

2.3 Software product line testing

Even though new testing challenges arise within the product line context, such as handling variability, combinatorial explosion of test and the trade-off between spending test effort in commonality testing versus variant testing (Paper IV), most traditional practices are still valid. Practices for testing object oriented systems and regression are especially useful in the product line context [92, 125]. Some general testing strategies may be particularly advantageous in, and adapted to, the software product line context, such as model based [94] and combinatorial [24] testing strategies.

Software product line testing can be described as a three-dimensional regression testing problem [117]. Developing a software product line, there are three main dimensions of software evolution to handle: 1) The hierarchical composition of the software system: the level of integration, 2) the evolution over time and 3) the derivation of variants. Figure 4 illustrates how testing is repeated over these three dimensions. The first dimension addresses the different phases of testing, described in Section 2.2, similar to a traditional V-model (e.g. unit testing, integration testing and system testing). At each of these levels, the test goals may be different, and typically correlated to the choice of test approach (e.g. functional test goals at unit level and non-functional at system level). The second dimension is the traditional regression testing perspective, described in Section 2.2, on which most research on regression testing is focused [137]. The third dimension is what makes the difference between software product line testing and traditional single system testing, the testing over variation in space. Variants may be derived by switching a parameter, changing a component or developing a new product from scratch (which is not the case in SPLE). If there are small internal differences between variants the testing may be performed with similar approaches as for regression testing. Basic testing trade-offs regard the balance of test effort over the different testing phases in a development project. Regression testing may be applied at all levels, and strategic decisions include: at which level, how often and how much need to be retested. In the SPL context, these decisions interact with the strategy on which variants to test, and to what extent.

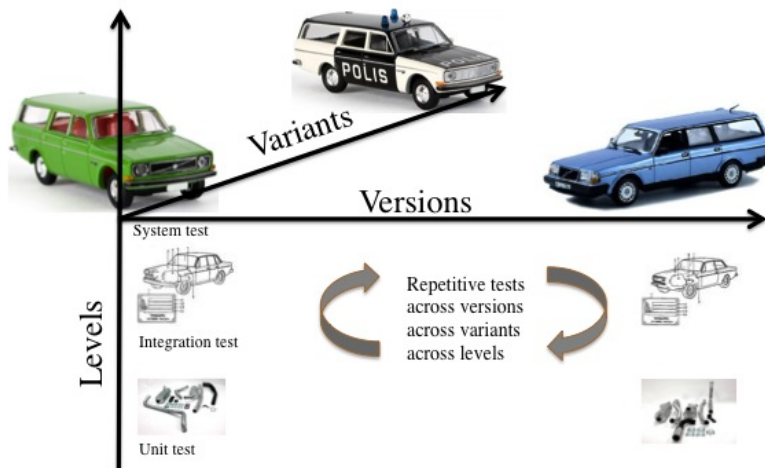


Figure 4: Overview of testing dimensions in software product line engineering.

This thesis focuses on decisions at an operational level, i.e. decisions on how to set the scope in a situation, with given constraints regarding cost and context, based on information from previous testing at other levels of the V-model, of earlier builds and other variants. Paper V investigates the complexity of these decisions through an in-depth case study in an emerging SPL organization and Paper VI proposes and evaluates a visual analytics strategy to support the decisions.

3 Research methodology

The main goal of this thesis work, similar to most research in software engineering, is of prescriptive character rather than descriptive, i.e. we want to solve a problem: How to improve software product line testing?, which would call for an applied research strategy. The following sections begin with a discussion of applied research in general, then continue with an overview of current focus and methodology in software engineering research and finally the process and methods used in this thesis work are discussed.

3.1 Applied research

van Aken [1] uses the term “design science” to distinguish between the research in applied sciences, such as medicine and engineering, from the “explanatory sciences”, such as physics and sociology, and “formal sciences”, such as philosophy and mathematics. The main differences between the three relate to the nature of

the outcome and to how the research is justified. Formal sciences build systems of propositions which are validated by internal consistency. Explanatory sciences describe, explain and predict observable phenomena with propositions which are accepted as true by the community based on provided evidence. Design science provide heuristic prescriptions for improvement problems. The context dependent nature of these prescriptions calls for pragmatic validation, i.e. it is impossible to conclusively prove its effects. Both explanatory and applied research rely on empirical methods to gain knowledge.

Finding a relevant solution to a problem requires an understanding of the problem to be solved, as well as an understanding of the effects of applying a proposed solution. Prescriptions need to be founded in both formal and explanatory theories. Many software engineering researchers claim a lack of relevant theory in software engineering [45] in general and there is room for more formal and explanatory research within this discipline despite its applied nature.

I have posted six research questions in this thesis, see Section 1.2, four of them are of explanatory type, while two have a design aspect, see Table 1. Explanatory questions may be of different kinds, depending on how much we already know about a specific topic or phenomenon: exploratory, descriptive, explanatory and predictive. Exploratory questions are useful in cases when a problem is not clearly defined. Descriptive questions may complement exploratory questions and aim at factually and systematically describing the characteristics of a problem. While exploratory questions are answered qualitatively, descriptive answers may also involve statistics. In this thesis exploratory and descriptive questions have been posted to better understand both the solution space (RQ1 and RQ2) and the application domain (RQ4 and RQ5). Explanatory questions follow the descriptive questions and seek to find casual relationships. In Table 1 I have classified the research questions in this thesis and the type of outcome corresponding with them.

3.2 Software engineering research

Software engineering research is an emerging discipline which has not yet established a commonly accepted pattern of methodology. Montesi et al. [86, 87] studied software engineering research by classifying and describing article types published in conferences and journals in the software engineering discipline and also in the discipline of biology and education. They found that methodology in software engineering research is not reported as frequently as in other disciplines. Instead, relevance for practice, novelty, and originality were the most important requirements for publication.

Many attempts to import research methodology guidelines from other domains to improve quality in software engineering research have been made: Kitchenham et al. [68] introduced guidelines for empirical research in software engineering based on guidelines from the medical research discipline. Kitchenham et al. have also provided guidelines for how to conduct experiments [68] as have Wohlin et

al. [133] based on social science practices. Easterbrook et al. [33] compare five classes of research methods which they believe are most relevant to software engineering: controlled experiments (including Quasi-Experiments); case studies (both exploratory and confirmatory); survey research; ethnographies and action research and provide guidelines for the choice of method. Runeson and Höst [118] provide a comprehensive guide to case study research in software engineering, compiled from methodology guidelines from other disciplines, mainly social sciences.

Along with the debate on quality of research and which methodology to use goes the debate on what focus software engineering research should have. Colin Potts [101] addressed 1993 the lack of problem focus in contrast to solution focus in software engineering research and argues for more of industry-as-laboratory research to balance the at the time more common research-then-transfer focus. This he argues would sacrifice revolution but gain steady evolution. The debate is still ongoing and the message in Briands' analysis of software engineering research today [15] is very much the same.

3.3 Research process and methods

Hevner describes a design science process including six steps [51, chapter 3.6]: 1) problem identification and motivation, 2) definition of the objectives for a solution, 3) design and development, 4) demonstration, 5) evaluation and 6) communication. Throughout the process knowledge is interchanged both with the knowledge base and the environment [50].

In this thesis the rationale behind the research was an intuitively sound proposal to use regression testing strategies to improve testing in a software product line context. However, regression testing in practice means something different from regression testing in research and I chose as a starting point for my research to achieve more knowledge about this gap, believing it be fundamental for the relevance of any solution proposal based on regression testing strategies. As a continuation I chose to focus on gaining knowledge about the SPL testing context.

The prescription in sight for this thesis work was a method to improve SPL testing. The initial proposal was rather vague, to use a regression testing approach, and the initial research task aimed at concretizing it. The design process in this thesis work included searches for applicable strategies in research (Paper I) and practice (Paper II) as well as identification of requirements in terms of SPLT challenges identified in literature (Paper IV) and practice (Paper V) and regression test challenges identified in practice (Papers II and III). No single existing technique was found to meet the identified challenges and the visual analytics proposal combines properties from several approaches. It was developed through experimentation with three different sets of authentic test data and evaluated in three different focus groups (Paper VI).

Table 1: Categorization of the research questions and activities reported in this thesis. All questions have been investigated with empirical methods. Four out of six questions (I, II, IV and V) are of explanatory type while two are of design type. The explanatory questions have been further categorized in: exploratory, descriptive and explanatory questions.

Question	Paper	Type of question	Type of research	Type of outcome
RQ1	I	Exploratory, Descriptive	Secondary re- search	Empirical knowledge
RQ2	II	Descriptive	Survey	Empirical knowledge
RQ3	I, II, III	Design	Action- research, Analytical	Heuristic prescription
RQ4	IV	Exploratory, Descriptive	Secondary re- search	Empirical knowledge
RQ5	V	Descriptive, Explanatory	Cases study	Empirical knowledge
RQ6	III, V, VI	Design	Cases study, Analytical, pragmatic validation	Heuristic prescription

Table 1 lists which methods I have used to retrieve answers for each research question. Details on how the different studies were carried out can be found in each paper respectively.

3.4 Validity

validity of each study is reported in each of the included papers. In this chapter I discuss the validity of the overall research process and outcomes in relation to the main goal: improving testing in a software product line context. I discuss validity from three different perspectives according to three-cycle model for design science [50]: Design, rigor and relevance.

The design cycle

Hevner and Catterjee model the design cycle as the heart of a design science research project [50]. The design cycle describes how the research iterates between the construction and evaluation of artifacts and its refinement based on this feedback. Here a balance must be maintained between construction work and evaluation work.

The research presented in this thesis includes a large share of knowledge seeking in relation to the novelty of the prescriptive outcome of the research. The benefits of information visualization to support decisions are not new knowledge [18,

124] even though it has not been applied in this context for this purpose before. This imbalance between construction work and evaluation work relates to the disorientation in the field of software engineering [45] as well as the gap between research and practice [15]. To build a relevant basis of knowledge this gap needs to be bridged. Although the prescriptions in this thesis are pragmatic and intuitive, they are novel both from a research and an industrial perspective. Theory on regression testing and SPL testing tend to overlook the complexity of the industrial context while practitioners do not have the tools to formalize it. Neither do researchers and practitioners speak the same language. Although not included in this thesis, the design process also involved industrial evaluations of regression testing strategies proposed in literature [36, 38]. None of these techniques were further refined by us to meet the SPL challenge but the experiences of the evaluations provided valuable insight to the problem and guided our design decisions towards the pragmatic prescriptions.

The prescriptions involve several elements which are derived from the exploratory, descriptive and explanatory studies. For example, the test coverage item model for data transformation is based on our observations of implicit testing goals in Paper V, and guidelines for regression test improvement derive from the survey on regression testing practice in Paper II and include the classification of techniques in Paper I. However, the displacement of focus towards explanatory research places the research somewhere between design science and explanatory science i.e. prescriptions are general and need to be further refined in the application context. In addition to general prescriptions, the outcome of those studies adds to the general knowledge base in software engineering.

Relevance

Design science should improve the environment in which it is applied. Relevance is assured if research activities address business needs [52]. The cost share of testing in software development [21] and increasing size and complexity of systems motivate the need for improved testing in general. Companies applying SPLE strategies report decreased development costs and shortened lead times [30, 79, 99]. It is a promising approach which is applied more and more to handle the complexity of variability and size in software systems. Testing strategies have to adapt to it. 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.

However, a prescription is not relevant per se if adapted to the SPLE but must be validated in the context. SPLE is to some extent an idealistic model of software development which is rarely fully implemented in industry. Hevner et al. define business needs as: *the goals, tasks, problems, and opportunities as they are perceived by people within the organization* [52]. Throughout this thesis work, collaboration with industrial partners have been central and all primary studies (Papers

II, III, V and VI) involve people within the organizations, through interviews, focus groups and surveys. None of the organizations represents a fully implemented SPL but all of them struggle with handling of variability and an exponentially growing testing task. Context is clearly reported from a product line perspective in all studies to enable analytical generalization [136] as far as possible.

The two main prescriptive outcomes of this thesis work are 1) a proposal on how to visualize historical test information to support test scoping decisions (Papers V and VI) and 2) a procedure guiding the practical adoption of regression testing evidence (Paper III). The visualization proposal was designed to meet the challenges identified in field and piloted on real data from three different test organizations. Furthermore it was tested and evaluated off-line by three focus groups of practitioners.

Thus, the visualization prescription is relevant in cases where large amounts of test cases are executed and there is a need to cut testing costs or to improve test efficiency, especially if there is a risk for test overlay due to organizational distribution of similar tests. The benefits of visualization is limited by the quality of the documented data. In all three cases to which the visualization was applied there were relevant data to visualize which could be extracted from databases or MS word documents. Relevance is also limited by the test process and strategy. To benefit from the visualization, the test process should involve repetitive testing and the test strategy allow for reduction of the test scope. The number of testing combinations in such variability intensive contexts is extensive and decision support for scoping is needed.

Rigor

Rigor relates to the way in which research is conducted [52]. It applies to every part of the research process: from our definition of questions, to the selection of and commitment to a method and to the interpretation and communication of results. In this thesis the above mentioned aspects of rigor have been carefully considered in all studies. Threats to validity are considered and reported in each study with respect to the construct of the study, reliability and internal and external validity [118, 133, 136]. Rigor in design science needs to be balanced with individuals' creativity and expertise in a domain. It defines the limit of produced knowledge and is closely related to relevance i.e. the level of detail in the abstract model of the gained knowledge should be sufficient but not too high. Limitations of the knowledge is here discussed in relation to the main contributions listed in section 1.2.

The main limitation of the two secondary studies (Paper I) and (Paper IV) is related to the scope of the studies and the quality of primary studies. In the systematic literature review on regression test selection the synthesis of results was inconclusive due to the diversity of techniques, goals and evaluation strategies among the primary studies. Still, the presentation of the results enables analytical

generalization and provides guidance for practitioners and researchers in selecting a regression test selection strategy and evaluating its effectiveness. However the scope of the study does not cover regression testing techniques other than selection techniques or empirical evaluations published later than 2006. How this limitation in scope affects the validity of the outcome depends on the recent activity in the field. Yoo and Harman surveyed the field of regression testing research until 2009 [137] and visualize some trends: 1) Interest in regression testing continues although it has decreased since 2007. 2) The scale of empirical studies seems to remain limited. 3) The share of selection papers decreases and instead the prioritization have been in focus the last years. In summary: practitioners can use the results of Paper I to get an understanding of the topic and a fairly representative assessment of the state of art in regression test selection but should also consider prioritization techniques as a possible means for regression test improvement.

The focus of the mapping study (Paper IV) is slightly different from the SLR since no synthesis of empirical results has been done. Instead the maturity of the research topic has been assessed and an overview of the research activity is provided. This overview is valid as long as the selection of primary research is representative and classifications are correct. The selection of primary research was carefully carried out using multiple search strategies. In addition both internal and external validation of the selected set of primary studies confirm its representativeness. Another similar study was conducted independently of ours and the two studies were compared to assess the reliability of the method [132]. This comparison highlights the lack of agreement in the community regarding classification of research types. Thus generalization of the conclusions should be made with care. However, generalization is supported through the transparency in the report regarding classifications. Our interpretation of the used classification scheme is described and the classification of each paper is explicit.

The other four studies (Papers II, III, V and VI) are carried out in close cooperation with industry which limits the knowledge in terms of *interpretation*, *representativeness* and *generalization*. *Interpretation* refers to the ability to have a common understanding of a phenomenon across different contexts, which could be both academia-industry and industry-industry. Especially in Papers II and V there are claims of contributing to increased understanding of the industrial context. In both cases a conceptual framework was developed in cooperation with the practitioners, introduced to all participants of the studies and reported in the final publications. *Representativeness* refers to how well the subject under study represents the object of study e.g. are the emerging product lines we have had access to good representatives for the real product line context or do the participants in the studies represent the populations in mind? Selection of subjects to study are based on a combination of availability and inclusiveness (some criteria need to be fulfilled), no attempts have been made to randomize samples. *Generalization* deals with the same questions but tackles them when drawing conclusions from a study rather than in the design phase. In all study reports careful case and context

descriptions are provided to support generalization. In addition, our own interpretations in the explanatory studies (Papers II and V) are conservative to minimize the risk of over interpretation of the data. Two studies provide prescriptive outcomes (Papers III and VI) regarding the regression test improvement procedure and regarding visualization as decision support. None of these prescriptions have been evaluated in the field but both derive from existing knowledge and both are evaluated off-line. In both cases prescriptions are general and thus apply to all relevant cases as described above. However, the set of relevant cases may include specific cases where other more specific prescriptions are applicable.

4 Research contributions

This section summarizes the contributions of the thesis, categorized according to the three sub goals stated in 1.1: 1) *Bridging the gap between regression testing state of art and state of practice*, 2) *Understanding the challenge of SPL testing* and 3) *Improving regression test scoping in the software product line context*. The relationships between the various contributions are illustrated in Figure 1.

4.1 Bridging the gap between regression testing state of art and state of practice

Papers I–III review the literature on regression test selection, surveys the state of practice and reports experiences from transferring empirical evidence into practice respectively.

A systematic review on regression test selection techniques

The main contributions of the systematic literature review (Paper I) are:

- A classification scheme for regression test selection techniques intended to make research results more accessible to practitioners within the field
- Overview and classification of regression test selection techniques which are evaluated in the literature
- Overview and qualitative analysis of reported evidence on regression test selection techniques
- Overview of metrics and strategies used for evaluation of regression test selection strategies

Several observations regarding relevance and applicability of evidence were made when analyzing the data of the systematic review. Firstly, most techniques are not evaluated sufficiently for practitioners to make decisions based on research

alone. Techniques are context dependent and in many cases evaluations are made only in one specific context. Few studies are replicated, and thus the possibility to draw conclusions based on variations in test context is limited. Moreover, often only one aspect is evaluated for a technique. Among the studies included in the systematic review, only 30% of the empirical evaluations considered both fault detection capabilities and cost reduction capabilities.

Secondly, standards for conducting empirical studies differ greatly just like the novelty of proposals. Some techniques could be regarded as novel at the time for their publications while others are variants of already existing techniques. In addition to this, some techniques are presented in a rather general manner, leaving much space for differences in implementation of a technique under study.

Thirdly, the evidence base is inconsistent. In cases where comparisons between techniques were possible, e.g. due to replications, there was not very strong evidence for differences between techniques and some results were even contradictory. To improve the situation there is a need for the research community to identify and focus studies on general regression testing concepts rather than on variants of specific techniques, to more systematically replicate studies in different contexts and gradually scale up the complexity and to identify and report important variation factors in the regression testing context.

A qualitative survey of regression testing practices

Paper II focus on industry practice of regression testing which is often based on experience, rather than systematic approaches. Through the means of focus group discussions and a questionnaire we wanted to identify challenges and good practices in industry. A total of 46 software engineers from 38 different organizations participated in the study. Results are qualitative and of great value in that they highlight relevant directions for future research. Results pinpoint the variation in regression testing situations and are summarized here under the main headings of: What?, When?, How? and Challenges?.

What? Regression testing involves repetitive tests and aims to verify that previously working software still works after changes to other parts. Focus can be either reexecution of test cases or retest of functionality. As for testing in general, the goal of the regression testing may differ between different organizations or parts of an organization. The goal may be either to find defects or to obtain a measure of the quality of the software. An additional goal may be to obtain a guide for further priorities in the project. Different kinds of changes to the system generate regression testing. Mentioned in the focus group and confirmed by the majority of the respondents were: *new versions, new configurations, corrections, changed solutions, new hardware, new platforms, new designs, and new interfaces*. The amount and frequency of regression testing is determined by the assessed risk,

the amount of new functionality, the amount of fixes, and the amount of available resources.

When? Regression testing is carried out at different levels (e.g., unit, integration, and system level) and at different stages of the development process. It was found that some organizations regression test as early as possible while other regression test as late as possible in the process, and some claimed that regression testing is continuously carried out throughout the whole development process.

How? Tests used for regression testing may be a selection of developer's tests, a selection of tester's tests, a selection of tests from a specific regression test suite, or new test cases are designed. Strategies for regression test selection included: complete retest, combine static and dynamic selection, complete retest of safety critical parts, select test cases concentrating on changes and possible side effects, ad hoc selection, smoke test, prioritize and run as many as possible, and focus on functional test cases. A project may include several different regression testing activities. Both manual and automatic regression testing are applied.

Challenges The focus group had an open discussion about both weaknesses and strengths in their regression testing practices. Some problems were common to most of the participants (e.g. lack of time and resources to regression test and insufficient tool support) while others were more specific. The main challenges relate to: *test suite maintenance* – much of the testing is redundant and there is a lack of traceability from tests to requirements; *test case selection* – it is hard to assess the impact of changes and to prioritize testing with respect to product risks and fault detection ability; *testability* – there is a need for design guidelines considering testability, modularization of the software, and clearer dependencies to ease regression test scoping; *the tradeoff between automated and manual regression testing* – automated regression testing causes problems, in terms of maintenance of test scripts, and manual testing is time and resource consuming; and *presentation and analysis of test results* – in many cases verdict reporting is inconsistent and often there is no time to do a thorough analysis. For each of the challenges covered in the discussion there were participants who had no solutions in place and participants who were satisfied with, and thus recommended, their solutions and tools.

A case study on regression test technology adoption

Paper III focus on the transfer of empirical evidence into practice. An industrial case study was carried out [36] to evaluate the possibility to improve regression testing in one test organization at Sony Ericsson Mobile Communications based on research results. In Paper III we analyze the procedure undertaken to identify the problem, search for relevant research and evaluate the effects of applying a

EBSE-approach	Automation-approach
<i>Focus on research answering the question which technique to use for the automation</i>	<i>Focus on practice answering the question what should be automated</i>
E1. Ask an answerable question	A1. Identify type of automation
E2. Find the best evidence	
E3. Critically appraise the evidence	
E4. Apply the evidence	A2. Identify level of automation A3. Apply primary evaluation criteria
E5. Evaluate performance	A4. Apply secondary evaluation criteria

Figure 5: Overview of the EBSE approach described by Dybå et al. [32] and the Automation approach described by Parasuraman et al. [95].

solution. Our analysis was based on frameworks from the evidence based software engineering (EBSE) paradigm [32] and automation literature [95]. The two models complement each other in their different foci, see Figure 5. While the focus of the EBSE model is on the assessment of empirical evidence, the automation model is focused on the practical effects of introducing the changes. The automation model distinguishes between different types and levels of automation and suggests primary and secondary evaluation criteria to be applied iteratively while adjusting the decisions on type and level of automation. The EBSE model is more general, i.e. it applies not only to automation but to any type of process improvement, and guides the procedure of matching real problems with empirical evidence as well as evaluating the quality of existing evidence and the performance of a suggested improvement.

Results from this case study pinpoint the need for more detailed methods for evaluating indirect effects of automation changes and matching the communication of empirical evidence with the description of the industrial context where automation is under consideration. Indirect effects may for example be skill degradation and overtrust in the tool. However, indirect effects may as well be positive such as increased consistency in documentation and cross learning between distributed teams. Our study provides examples of how to apply existing general guidelines as well as of how to create subarea specific guidelines for the different tasks of the automation procedure. The results are here summarized related to the three main tasks of the regression test improvement procedure as defined in Figure 5: identification of preconditions (E1 and A1), identification of relevant research (E2-3) and evaluation of effects (E4-5 and A2-4).

Identification of preconditions: Our case study demonstrates the benefits of using structured methods and frameworks to define the scope and effect targets of the improvement as well as the need for even more detailed support for identifying which trade-offs and context constraints need to be considered in the search for a solution. We used interviews to gather information on the problem which helped us extract the meaning of “improving regression testing” in the current context into a list of four concrete effect targets. Furthermore, the context information was structured according to an existing framework [97] for context description in software engineering. This general framework offered some support in identifying relevant context constraints for the selection of technique, identifying indirect effects of the automation change, and communicating the results of the evaluation. However, it did not support the more detailed search and comparison of regression testing techniques on the same level of abstraction as they are reported in literature.

Identification of relevant research: The search for relevant techniques should be driven by scope of automation, effect targets and context constraints. Our case study pinpoints the mismatch between empirical evidence and industrial needs. Effect targets identified in our case study correlate with general needs in industry and are partly addressed by research. However, constraints in the industrial context and current practices delimited the selection considerably.

Evaluation of effects: Automation changes may have both direct and indirect effects. Direct effects may be evaluated against the effect targets; in the case of regression testing detailed evaluation frameworks for evaluating cost and efficiency exist [111, 115]. Indirect effects involve effects of the changed, added and removed tasks caused by the automation change. These effects are harder to evaluate and no specific guidelines were available for that purpose. Our case study exemplifies the application of existing general frameworks as well as the development of context specific checklists to support analytical evaluation of indirect effects.

4.2 Understanding the challenge of SPL testing

Papers IV–V review the literature on software product line testing, study software product line testing in practice through an in depth case study and models the testing tradeoffs needed in such a context.

A systematic mapping study on software product line testing

The main contribution of the systematic mapping of research on software product line testing (Paper IV) are:

- Overview and classification of research on software product line testing

- Overview and quantitative analysis regarding type of research, type of contribution and focus of research

The main focus of the research in software product line testing is the vast amount of possible tests that may be run. Research proposals deal with questions on how to handle the variability, the combinatorial explosion of tests, and how to balance test efforts between testing of common components and the configured product variants, as well as between different phases of testing. Research topics include: *test organization and process*, *test management*, *testability*, *test approaches at different levels of tests* and *test automation*.

Test organization and process: Research on test organization and process focus on the testing framework, seeking answers to how the testing activities and test assets should be mapped to the overall product line development, and also how product line testing should be organized overall. McGregor points out the need for a well-designed process and discusses the complex relationships between components and products, and different versions of both components and products [82]. He argues for a structure of test assets and documentation in alignment with the structure of the constructed products. Pohl et al. present four principal strategies for the trade-off between domain and application testing [99]: P1) test everything at the domain level, P2) test everything at application level, P3) test a sample at domain level, and the full application testing and P4) test common part at domain level and variability at application level.

Tevanlinna et al. [125] addressed the problem of dividing product line testing into two distinct instantiations of the V-model, and concluded that testing is product oriented and no efficient techniques for domain testing exist. Instead they distinguish between four different test reuse strategies: R1) product by product testing (no reuse), similar to P2; R2) incremental testing, test at application level what is changed between the current product and previously tested products; R3) reusable assets instantiation, focus testing on the domain level, similar to P4; and R4) division of responsibility – testing is seen from an organizational point of view.

Test management: Research on SPL test management includes test planning and assessment, fault prediction, selection of test approaches, estimates of the extent of testing and test coverage. *Test planning frameworks:* Tevanlinna [125] presents a tool called RITA (fRamework Integration and Testing Application) to support testing of product lines. Kolb presents a conceptual proposal for risk-based test planning and test case design [69] and McGregor and Im outline a conceptual proposal that address the fact that product lines vary both in time and space [84]. *Cost/benefit assessment models:* Zeng et al. identify factors that influence SPL testing effort, and propose cost models accordingly [139]. Jaring et al. propose a model, VTIM (Variability and Testability Interaction Model) to support man-

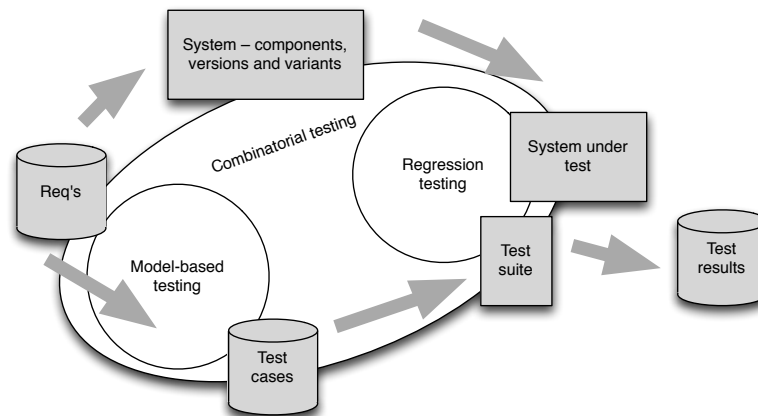


Figure 6: Scope of combinatorial testing, model-based testing and regression testing.

agement of trade-offs on the binding point for a product line instance [58]. They illustrate the model on a large-scale industrial system. *Test approaches:* Three different test approaches seem to be particularly beneficial for testing a software product line: *combinatorial testing approaches*, *model-based testing approaches* and *regression testing approaches* [24, 83, 88, 125]. Figure 6 illustrates the scope of the three approaches. Model-based testing strategies focus on the creation of test cases and seeks to automatically generate test cases from explicit test models. Regression testing strategies focus on the selection of existing test cases for execution based on analysis of test cases and test objects. Combinatorial strategies deals with both creation and selection of test cases.

Testability: McGregor discusses testability, referring to characteristics of the software products that help testing [82]. Trew identifies classes of faults that cannot be detected by testing and claim the need for design policies to ensure testability of an SPL [127]. Kolb and Muthig discuss the relationship between testability and SPL architecture and propose to improve and evaluate testability [70].

Test approaches at different levels of test: Most research is spent on system and acceptance testing and the most frequent goal is automatic generation of test cases from requirements. Requirements may be model based, mostly on use cases [106], formal specifications [85], or written in natural language [6]. *system and acceptance testing:* Hartmann et.al present an approach based on existing UML based tools and methods [48]. Reuys et al. define the ScenTED approach

to generate test cases from UML models [107], which is further presented by Pohl and Metzger [100]. Bertolino and Gnesi introduce PLUTO, product line use case test optimization [9–11]. Bashardoust-Tajali and Corriveau extract tests for product testing, based on a domain model, expressed as generative contracts [6]. Geppert et al. present a decision model for acceptance testing, based on decision trees [41]. The approach was evaluated on a part of an industrial SPL. Li et al. utilize the information in execution traces to reduce test execution of each product of the SPL [77]. *Integration testing*: The ScenTED method is proposed also for integration testing in addition to system and acceptance testing [109]. Reis et al. specifically validate its use for integration testing in an experimental evaluation [106]. Kishi and Noda propose an integration testing technique based on test scenarios, utilizing model checking techniques [67]. Li et al. generate integration test from unit tests, illustrated in an industrial case study [78]. *Unit testing*: Different approaches to create test cases based on requirements including variabilities are proposed with a focus on how to cover possible scenarios. In ScenTED [108], UML-activity diagrams are used to represent all possible scenarios. Nebut et al. [89] use parameterized use cases as contracts on which testing coverage criteria may be applied. Feng et al. use an aspect-oriented approach to generate unit tests [39].

A case study on test overlay in an emerging software product line

Paper V studies one aspect of software product line testing in depth, test overlay, through the means of an industrial case study. The studied case is the testing in the case company's development of Android¹ 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.

The focus in this study is on manual testing of functional and quality requirements. This testing is characterized by a higher degree of creativity for the tester and less detailed documentation. Relations between different test executions and implicit testing goals are identified and expressed in terms of test coverage items (TCI:s)², capturing different levels of detail as well as different purposes of tests.

The case context The product line products, which were developed in a software project in the case study context, comprise different product variants (about 10), called *product configurations* in Figure 7. The products are in turn customized for a number of different customers and market segments (hundreds) which have

¹<http://www.android.com/>

²In Paper V the term used is only "coverage items".

different software requirements, and are called *release configurations*. Each release is packaged in a *product package*.

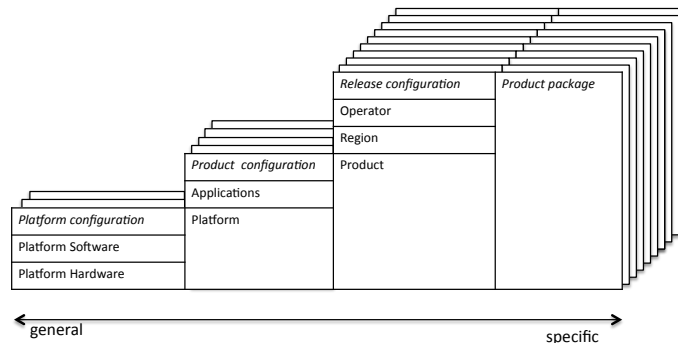


Figure 7: Configuration view of the product line under study.

The software development is distributed over three organizational units (*core software*, *application software* and *product composition*), where they primarily focus on platform, product and release configurations, respectively, as defined in Figure 7. 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. Parts of the organizations are distributed over three continents.

Testing in the core software and application software organizations are conducted at (at least) three main levels of test abstraction each, which involves repeated testing of common parts. *Feature tests* (unit testing, structural and functional testing) are carried out by the functional area teams. *Integration tests and system test* are carried out at both the test departments for platform and product. 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.

Overlaid testing In the case, the amount of *overlaid* testing, and the factors causing overlay, were investigated. The analysis distinguishes between *test design overlay* and *test execution overlay*. Test design overlay refers to the existence of more than one test case that test the same TCI for the same purpose. Test execution overlay refers to multiple executions of the same TCI with the same purpose. Two of the research questions studied regard the industrial SPL testing context:

1. *How much testing in a variability-intensive context is overlaid and which is redundant?* – Testing is repeated across abstraction levels, versions and variants which could imply that multiple testing is done on the same TCI:s.
2. *When and why does overlaid testing appear?* – If overlaid testing exist, which factors are causing the overlaid testing?

In total 517 test executions of 192 *different* test cases were identified, which tested the case function, over a period of 22 weeks. The failure identification rate is generally low: 15 of the 517 test executions failed. Feature testing and integration testing run only one failing execution each.

Overlay in test design At the highest level of coverage item abstraction, these 192 test cases cover 19 unique TCI:s. Hence, 90% of the test cases could be considered overlaid since they do not cover any unique coverage items. A large share of the design overlay identified at the highest level of abstraction 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 to test the function in terms of compatibility with different models of devices, and types and sizes of memory cards. Increasing the detail level of the ‘purpose’ parameter, there is no overlay between the different test levels: feature integration and system testing, and no overlay within feature testing. There is still design redundancy within integration testing and system testing at this level, 33% and 63% respectively.

Overlay in test execution Overlay in test execution is defined as re-execution of a coverage item and could origin both in overlay in the test design or the re-execution of a test case. However the overlaid test execution is not redundant, if it has been affected by a change since the last execution. At the highest level of abstraction, 517 test executions tested the case function. 96% of these are overlaid. The overlay remains high even for analysis at lower abstraction levels.

Factors causing redundancy Based on the analysis of the test data and interviews with nine testers and managers, three key factors causing test redundancy were identified:

1. Distribution of test responsibilities – Organizational distribution had more impact than geographical.
2. Inconsistent documentation of test cases – Importance of consistency in design and documentation of test cases seem 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 redundancy in the testing.
3. 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.

Behind these factors, there is a chain of causes and effects, described in Figure 8, which is elaborated in depth in Paper V.

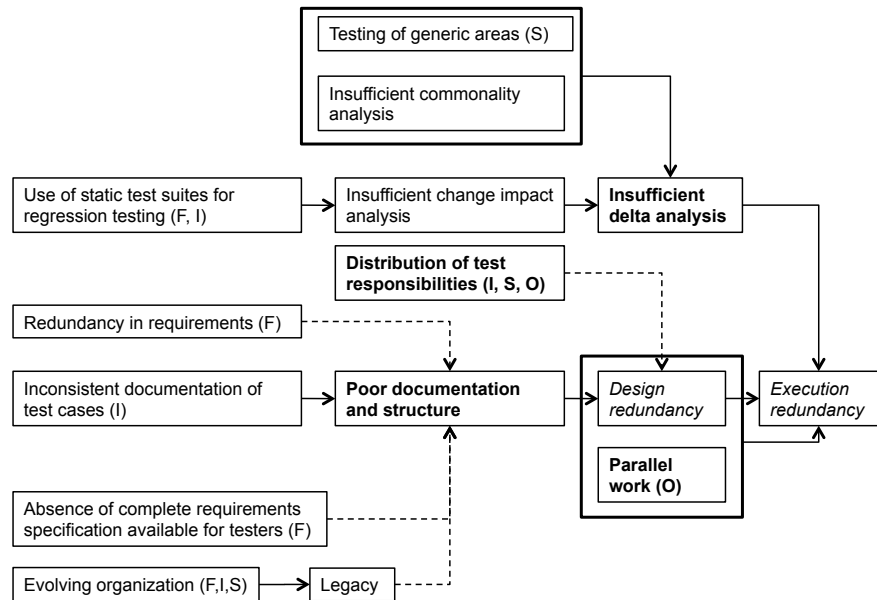


Figure 8: Graph illustrating the hypotheses derived from the case study. Dashed arrows indicate relations partly contradicted by the quantitative data. F = Feature test, I = Integration test, S = System test, O = Overlap.

4.3 Improving regression test scoping in the software product line context

Papers V and VI propose and evaluate a visual analytics strategy which aims to address the identified challenges. A tool for decision support seeks to automate the management parts of the testing chain, illustrated in the left semicircle in Figure 3 in Section 2.2. Most of the RT techniques offered in literature provides such support on a rather high automation level, i.e. with a low level of user interaction, which limits their flexibility and make them very context dependent. Since our industrial studies all highlight the complexity and variation in regression testing situations we propose a pragmatic tool offering support at a lower level of automation, i.e. allowing for more user interaction. Furthermore, previous studies on SE technology adoption observe a status quo mindset [31,62], meaning that organizations are reluctant to make radical changes to existing process. Thus, we propose a tool that may adapt to and evolve incrementally within the complex context. The first step is just to make the existing practices transparent.

A visualization proposal

The third research question in Paper V regards the possibility to improve regression testing within the industrial SPL testing context:

3. *How can visualization support test scoping decisions?* – We assume that visualization is a powerful tool when handling large volumes of data, which is the case for SPL testing. Thus it is relevant to study prerequisites for visualization in this context.

Based on the observed needs for improvement in the case study and our experience of analyzing overlay within this context, we draw conclusions about how to provide a visualization that is correct, understandable and relevant for the purpose of supporting test scoping decisions in a SPL context. The visualization should include: test design coverage, test execution progress and priorities of both TCI:s and variants. Furthermore the visualization tool need to be flexible and allow for different perspectives on the testing due to different testing goals at different stages of a project as well as the different roles in the organization.

Empirical evaluation of the visualization proposal

Paper VI reports our experiences of implementing and evaluating visual analytics for supporting test evaluation and planning. A set of various visualization prototypes were created to illustrate the concepts of our visualization proposal. Data from three different real world software contexts were extracted and transformed into data tables in various ways. The data tables were in turn mapped to visual structures and visualized with the MosaiCode tool [81], see Figure 9 which was originally developed for software visualization. For each of the three software contexts we set up a focus group meeting with test managers who used the visualization prototypes for performing a set of simulated evaluation and planning tasks. Meanwhile, we observed how they used the tool and held a structured discussion about their experience of the tool put in relation to their daily work.

Results from this study were analyzed according to the visualization reference model by Card et al. [18] which describes visualization as a set of adjustable mappings from data to visual form: *data transformation*, from raw data to data tables; *visual mapping*, from data tables to visual structures; and *view transformation* from visual structures to views. *User interaction* may control different parameters of these mappings.

All study participants confirmed potential usefulness of the visual analytics for evaluating and planning testing as well as for communicating decisions to managers and subordinate testers. We focused on visualizing test execution history and extracted data from test management databases and from natural language test reports in MS Word format. The extracted data was transformed into data tables with test coverage items, TCI:s, described by a set of dimension values and attribute values. Although, all participants found this visualization useful for assess-

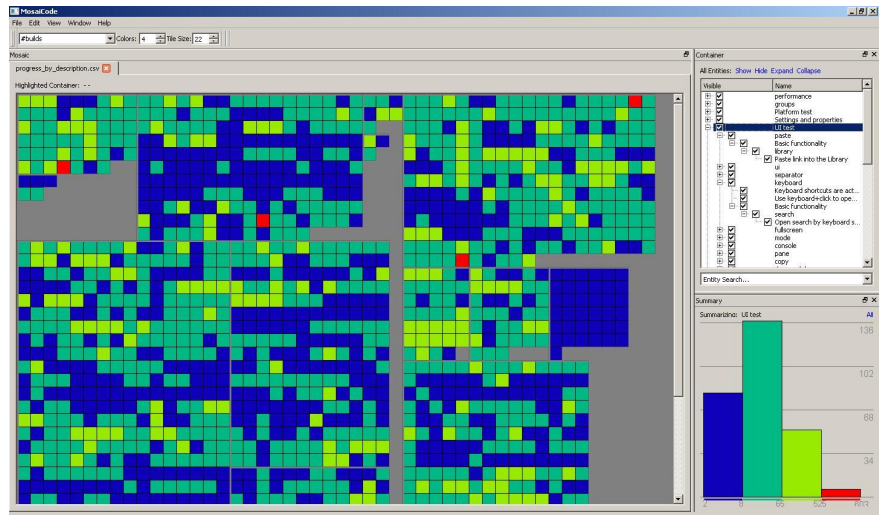


Figure 9: Screenshot of one of the prototype visualizations – Tiles in the Mosaic window represent the about 1500 different test cases that have been executed in the Firefox regression test suite. Information from the about 300 000 test executions is aggregated and a number of different attribute values are visualized through the color schemes of the tiles and in the histogram.

ing test coverage and making decisions, some pinpointed the need for additional information to make informed decisions. Furthermore, our evaluation showed the need to visualize several dimensions of the coverage items. In Table 2, two example TCI:s defined in 6 dimensions are presented. Different views were needed for different types of tasks. This was also true for the derived attribute values; all proposed metrics were useful but for different tasks.

The main components of a visual structure are the use of space, the marks, and the graphical properties of the marks [18]. In our visualization prototype we used spatial position to encode dimension values of TCIs in the mosaic window and colors of the tiles to encode attribute values. The mosaic view showed to be useful to the one being new to a test scope or when the data set is large, as it provided a good overview of both test design coverage and execution progress. The container view was useful to the one being familiar with the scope and the structure, for navigating.

For the planning task more user interaction was requested. Especially, the participants requested an ability to filter data on different criteria and to manually map colors to ranges of attribute values. The MosaiCode tool provided interactive view transformation in terms of zooming, control of tile size and number of different colors on tiles. Practitioners considered the zooming function very useful but preferred automatic setting of tile sizes.

Table 2: Data table of example test coverage items – The hierarchical positions are described along six dimensions of the testing strategy. TCI 11 is here a sub TCI of TCI 1021 with respect to the time dimension. The abstraction level of the TCI determines the derived values of the list of attributes.

	TCI 1021	TCI 11
Dimension: Functionality	System/Social phonebook/Add contact	System/Social phonebook/Add contact
Dimension: Purpose	All/Interaction/Incoming message	All/Interaction/Incoming message
Dimension: Variant	All/Blue star	All/Blue star
Dimension: Platform	All	All
Dimension: Organization	OrganizationA/Application software/System integration test	OrganizationA/Application software/System integration test
Dimension: Time	All	All/2012/w46
Attribute: #executions	13	0
Attribute: #failures	0	0
Attribute: #days since last execution	30	30
...

5 Related work

This chapter points out current research which is directly related to the research contributions presented in this thesis. A general positioning of the research is provided in the introduction, Chapter 1. Background information on the main areas under study (i.e. software product line engineering, software testing and software product line testing) is provided in Chapter 2.

5.1 Related publications by the author

In addition to the papers included in this thesis the research project has generated a number of related publications.

In our attempt to synthesize research on regression test selection (Paper 1) and provide advice for practitioners we discovered some shortcomings in the evidence base. In [119] we elaborate on the problem of transferring research into practice and give proposals regarding test technique benchmarking in terms of context applicability.

A Brazilian research group carried out a mapping study [25] similar to ours (Paper 4). Unaware of each other we chose the same topic and research method and conducted our studies at about the same time. Both studies were published and we summarize our results in [27]. This coincidence also gave us the opportunity to conduct a meta study on the reliability of systematic mapping studies in software engineering [132].

We have carried out two real life evaluations of two different regression test approaches which are reported in [38] and [36]. In both cases we selected techniques based on the needs and constraints in the industrial contexts. Both evaluations showed promising results and provided valuable insights on the challenge of transferring research into practice, which were summarized in Paper III.

We also conducted an extensive interview study, investigating the large scale industrial product line context from the perspective of aligning requirements and test [13, 120]. We revealed a number of challenges related to eight different problem areas: organization and processes, people, tools, requirements process, testing process, change management, traceability, and measurement. This study was extended with interviews from additionally five companies and an analysis covering both challenges and practices used to meet the identified challenges [13].

In [35] we discuss the need and challenges of providing decision support for test planning and test selection in a product line context, and highlights possible paths towards a pragmatic implementation of context-specific decision support of various levels of automation.

We were also invited to write a chapter, for the book series "Advances in Computers", presenting an overview of the subject regression testing in software product line engineering [116]. This series targets a broad audience and the level of presentation is aimed at readers who are generally not specialists in the field of software testing.

5.2 Other researchers' work

Regression testing

Regression testing research focus on how to reuse existing tests as testing is repeated over time. Three main groups of approaches have been studied: Test suite minimization, test case selection and test case prioritization. Yoo and Harman survey the literature within each of these areas and provide an overview of the state of research [137]. In order to determine which technique is appropriate for a certain situation, several factors need to be considered such as which *input* does the technique require? on what *level of abstraction* is the analysis made?, and what *empirical support* is there for these claims?

Input: Most of the proposed techniques are code based, conducting analysis on source code (e.g. the selection techniques by Leung and White [75], Chen et al. [20]), intermediate code (e.g. the selection techniques by Rothermel et al. [113], White and Abdullah [129]), or binary code (e.g. the selection technique by Zheng et al. [140]). Similarly most minimization techniques optimize the structural coverage of a set of tests (e.g. work by Horgan and London [53], Harrold et al. [47], Offutt et al. [91]). Prioritization of test cases may be based on different criteria of which code coverage is one (e.g., by Elbaum et al. [34]). Some regression testing techniques are based on a certain type of specifications or models (e.g., the selection technique by Sajeew and Wibowo [121], the minimization techniques by

Harder et al. [46], Korel et al. [72], and the prioritization technique by Korel et al. [71]). There are also some recent examples of both selection and prioritization techniques based on project data such as failure reports or execution history of a test case [38, 66]. Other criteria for test case prioritization are, e.g., combinatorial. [16, 23] or human based [126, 138].

Level of abstraction: While most of the proposed techniques conduct analysis at a low level, e.g. statements, there are other examples, e.g. a class based technique by Orso et al. [93] or component based by Sajeev and Wibowo [121] and Zheng et al. [140].

Empirical support: Several empirical evaluations of regression testing techniques have been made lately. Our systematic review (Paper I) and the survey by Yoo and Harman [137] provide good overviews. However, as concluded in Paper I, the evidence base is incomplete since the techniques are context dependent and in many cases evaluations are made only in one specific context.

Regression testing in the SPL context

As defined in [117] the SPLT problem can be described as a 3D regression testing problem (over levels, versions and variants). Also McGregor and Im outline a conceptual proposal that address the fact that product lines vary both in time and space [84]. Traditionally regression testing strategies deal with the repetitive testing over several versions of the software, the time dimension. Some attempts have been made to apply these also to the space dimension: covering arrays [102, 135], architectural diagrams [26] based on formal models. Stricker et al. propose a data-flow based technique to avoid redundant testing in application engineering [123]. Kim et al. introduce the idea of shared executions to avoid redundancy [65] Their idea builds on the same principle as the TCI model described in this thesis but on a lower level of detail. Long et al. [80] demonstrate synergies and overlap in code coverage between what they call loosely-coupled software development communities, which also applies to SPL development.

Nörenberg et al. [90] propose a lightweight approach for regression test selection which they argue applies to the software product line testing problem as well. They validate their approach on an embedded system in the automotive domain. The approach is based on system requirements and their association with test cases. However, although lightweight, the approach requires the existence of a complete system specification containing information on dependencies between system functions and contributing components. In addition to a standardized system specification the approach also requires a standardized testing strategy within the organization to ensure an equal level of available information within test specifications.

On transferring research in SE into practice

Evidence based software engineering (EBSE) is a paradigm, inspired by medical practice, trying to support the application of empirical evidence in practice. One central tool is systematic literature reviews. Dyba et al. propose a systematic EBSE approach for a practitioner which involves five steps [32] to support and improve decisions on technology adoption. The focus of their approach is the identification and appraisal of evidence. A model to support the evaluation of rigor and relevance of technology evolutions in software engineering was proposed by Ivarsson and Gorschek [57].

Rainer et al. [105] investigate the discrepancy between software engineering practices and the EBSE paradigm. They conclude that practitioners need to be persuaded to adopt EBSE practices and that empirical evidence per se is not important in the initial persuasion strategy. They further evaluate the use of the EBSE guidelines by undergraduate students empirically [103, 104] and find that EBSE is very challenging for non-researchers and that rigorous and relevant evidence may not exist to answer their questions. Gorschek et al. [42] proposed another model to support technology transfer focusing on the collaboration between researchers and practitioners.

Specifically, the gap between research and practice in the regression testing field is analyzed by Nörenberg et al. [90]. They highlight the imbalance between different evaluation criteria in empirical evaluations of regression testing techniques and call for more focus on generality and efficiency as defined by Rothermel and Harrold [111]. The use of various research methodologies to improve the relevance of software engineering research have also been proposed: ethnographies [54], case studies [118] and action research [3].

Visualization in SE

Visualization helps reducing cognitive complexity [18]. Software visualization is applied in various areas: e.g. static and dynamic code visualization [4], fault diagnostics and requirements analysis [43]. A survey of software maintenance, reengineering and reverse engineering [73] shows that 82% of researchers consider software visualization important or necessary for their work. Another survey with industry practitioners [7] identifies benefits of software visualization tools: reducing costs, increasing comprehension of software, increasing productivity and quality, management of complexity, and finding errors [28]. Kienle and Muller identify requirements for software visualization tools with a focus on tools targeting the domain of software maintenance, reengineering, and reverse engineering [64].

Diehl provides an overview of the area of software visualization defined as “*the visualization of artifacts related to software and its development process*” [28]. He identifies three main groups of techniques: 1) visualizing static program properties, dynamic program properties and software evolution. Diehl compares differ-

ent visualization methodologies [29] and apply visualization techniques to explore rules extracted from historical software information [17].

Pleuss et al. [98] provide a comparative overview of visualization techniques to support product configuration in software product line engineering. Jones et al. [60] show how visualization of test information may support fault localization. Araya presents a visualization tool HaPAO to visually assess the quality of code coverage of unit tests [2]. A range of metrics for each class are visualized to illustrate how the methods in it have been covered during a set of tests.

6 Future research directions

In the first part of this thesis work, reported in Papers I–III, I observe some challenges in matching research on regression testing with the industrial need for improvement of the same. These challenges originate in communication gaps between researchers and practitioners as well as lack of general knowledge on regression testing concepts. In this thesis I provide an increased understanding of both the research and the industrial needs to overcome the communication gap. I also provide an extended model for technology adoption to guide practitioners in adopting research results and introduce tool support into a process. The second part of the thesis (Papers IV and V) focus on the challenges for regression testing in the software product line context. Support is needed for managing huge amounts of tests repeatedly executed on similar software due to variability in the software, iterative development and complex organizational structures. Finally, in Papers V and VI, I propose and evaluate visual analytics of historic test execution data as a means to meet those challenges. This section describes ways to advance this research further in terms of *1) improving the accessibility of regression testing research, 2) increasing the knowledge on general regression testing concepts, 3) providing support for evaluating indirect effects of introducing regression testing tools to a process and 4) improving and evaluating the visualization approach in different contexts.*

Improve accessibility of regression testing research. There are several reasons why regression testing research is hard to access and implement in industry. Except for the large variation among researchers in how regression testing strategies are described and evaluated, researchers and practitioners approach the regression testing problem at different abstraction levels, from different perspectives and to some extent with different effect targets. Our synthesis of empirical results and classification of evaluated techniques approach the first obstacle and abstracts the current knowledge one level. This supports practitioners in getting an overview of existing knowledge in the field. In addition they need guidelines supporting the identification of regression testing problems in their own context. There may be a range of factors causing inefficiency in regression testing that do

not originate from the regression testing itself. Thus support is needed to extract the regression testing problem. To search for applicable techniques practitioners would also need a framework for describing context factors which are crucial for the selection of regression testing strategies. Similarly such framework should support researchers in classifying and reporting evaluation context of regression testing studies. Such framework should preferably build on secondary studies of both regression testing research and industrial regression testing context.

Increase knowledge on general regression testing concepts. While several secondary studies on regression testing research are already available, this is not the case for context studies. There is a need for more studies focusing on the industrial context with respect to regression testing. Secondary studies could include both primary studies reporting industrial contexts although it is not their focus, and less rigorous experience reports. Such attempts have been made in the software product line testing case [59]. Regarding the evidence base on regression testing strategies, available secondary studies [37, 137] are of course limited as well since they depend on available primary studies. These studies do however pinpoint the gaps and stress the need for more evaluation of general regression testing assumptions rather than specific implementations of techniques.

Provide support for evaluating indirect effects of introducing regression testing tools to a process. In Paper III we introduce a model for how to introduce testing process changes considering both available research and pragmatic effects of the change. However, although we used it for evaluating the effects of regression testing improvement in a real industrial case, the model is rather general and needs to be complemented with more subarea specific guidelines. Such guidelines may evolve from additional real life evaluations of regression test improvement actions.

Improve and evaluate the visualization approach in different contexts. In Paper VI we discuss and evaluate our visualization approach. It was appreciated by practitioners from different industrial contexts as a way to bring order in a complex regression testing environment and meet the SPLT challenge to assess test quality and plan testing when large amounts of repeated tests are needed. They also provided valuable feedback on how to improve the visualization further. However, as discussed throughout this thesis, such improvements and further evaluations need to be made incrementally within a specific context. Although we have a lot to learn from each other's experiences, no general solutions exist and researchers in software engineering need to support practitioners and computer scientists in bringing clarity in the effects of applying specific treatments to specific contexts. Two examples of technical aspects of the visualization to refine and evaluate further are the cluster analysis and the correlation between different attribute

values and fault detection abilities. Other aspects to evaluate are the indirect and direct effects of adopting such tool in an organization, the effects of providing different types of human interaction as well as different ways of visualizing several dimensions of the testing.

7 Conclusion

Industry practice of regression testing does not utilize the knowledge gained in research. There may be several reasons for that. Test maturity in an organization may be generally low, leading to shortsighted test planning with too short lead times for test. A lack of testing resources in combination with short lead times prevents both the use of systematic approaches and improvement work itself.

Even with a more mature test organization and dedicated resources for test improvements it is hard for a practitioner to identify relevant and applicable research. Many proposed regression testing techniques solve a very specific problem in a similarly specific context. For a tester operating in a complex industrial context it might not make sense to isolate a point of improvement to match those proposals.

Large software organizations with a product line development approach face many challenges regarding testing. Trade-offs need to be done along three dimensions of repeated testing and consider a range of goals and constraints. Furthermore, test management is a part of a large management system, including release planning, requirements management, configuration management, development management, test management, and more, which makes it difficult to enroll major changes from any of those single perspectives. If test maturity in an organization is low, testing may be considered the last instance of development implying that testing aspects have low priority in decisions on for example which development paradigm to use, how to specify requirements or design the system.

Test planning support, like any decision support system, must operate in a complex context and need to be pragmatic, adapted to the context and evolve incrementally within the context. Also introducing automated support for regression test scoping need to be done in increments including both evaluation and decision tasks regarding for example: identification of preconditions, evaluation of the effects of automation changes and implementing, evaluating and improving the implementation.

A first step towards decision support is the visualization of test execution data as proposed in this thesis. Through visualization of relevant information at a proper level of detail, test management in general may be supported. Such tool would also provide a framework which enables research based and context specific regression testing improvements.

REFERENCES

- [1] Joan E. van Aken. Management research based on the paradigm of the design sciences: The quest for field-tested and grounded technological rules. *Journal of Management Studies*, 41(2):219–246, 2004.
- [2] Vanessa Peña Araya. Test blueprint: an effective visual support for test coverage. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE '11)*, pages 1140–1142, New York, NY, USA, 2011. ACM.
- [3] David E. Avison, Francis Lau, Michael D. Myers, and Peter Axel Nielsen. Action research. *Communications of the ACM*, 42(1):94–97, January 1999.
- [4] Thomas A. Ball and Stephen G. Eick. Software visualization in the large. *Computer*, 29(4):33–43, April 1996.
- [5] Ghinwa Baradhi and Nashat Mansour. A comparative study of five regression testing algorithms. In *Proceedings of the Australian Software Engineering Conference (ASWEC '97)*, pages 174–182, 1997.
- [6] Soheila Bashardoust-Tajali and Jean-Pierre Corriveau. On extracting tests from a testable model in the context of domain engineering. In *Proceedings of the IEEE 13th International Conference on Engineering of Complex Computer Systems (ICECCS 2008)*, pages 98–107, April 2008.
- [7] Sarita Bassil and Rudolf K. Keller. Software visualization tools: survey and analysis. In *Proceedings of the 9th International Workshop on Program Comprehension (IWPC 2001)*, pages 7–17, 2001.
- [8] Stefan Berner, Roland Weber, and Rudolf K. Keller. Observations and lessons learned from automated testing. In *Proceedings of the 27th international conference on Software engineering (ICSE '05)*, pages 571–579, 2005.
- [9] Antonia Bertolino, Alessandro Fantechi, Stefania Gnesi, and Giuseppe Lami. Product line use cases: Scenario-based specification and testing of

- requirements. In *Software Product Lines - Research Issues in Engineering and Management*, pages 425–445. Springer-Verlag, 2006.
- [10] Antonia Bertolino and Stefania Gnesi. Use case-based testing of product lines. In *Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering (ESEC/FSE-11)*, pages 355–358, 2003.
- [11] Antonia Bertolino and Stefania Gnesi. PLUTO: a test methodology for product families. In *Software Product Family Engineering Software Product Family Engineering*, volume 3014 of *Lecture Notes in Computer Science*, pages 181–197. Springer-Verlag, 2004.
- [12] John Bible, Gregg Rothermel, and David S. Rosenblum. A comparative study of coarse- and fine-grained safe regression test-selection techniques. *ACM Transactions on Software Engineering and Methodology*, 10(2):149–183, 2001.
- [13] Elizabeth Bjarnason, Per Runeson, Markus Borg, Michael Unterkalmsteiner, Emelie Engström, Björn Regnell, Giedre Sabaliauskaite, Annabella Loconsole, Tony Gorschek, and Robert Feldt. Challenges and practices in aligning requirements with verification and validation: A case study of six companies. Manuscript submitted for publication.
- [14] Jan Bosch. *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. Addison-Wesley, 2000.
- [15] Lionel Briand. Embracing the engineering side of software engineering. *IEEE Software*, 29(4):96–96, July 2012.
- [16] Renée C. Bryce and Charles J. Colbourn. Prioritized interaction testing for pair-wise coverage with seeding and constraints. *Information and Software Technology*, 48(10):960–970, 2006.
- [17] Michael Burch, Stephan Diehl, and Peter Weißgerber. Visual data mining in software archives. In *Proceedings of the ACM symposium on Software visualization (SoftVis '05)*, pages 37–46, New York, NY, USA, 2005.
- [18] Stuart K. Card, Jock Mackinlay, and Ben Shneiderman, editors. *Readings in Information Visualization: Using Vision to Think*. Academic Press, 1 edition, February 1999.
- [19] Emanuela G. Cartaxo, Patrícia D. L. Machado, and Francisco G. Oliveira Neto. On the use of a similarity function for test case selection in the context of model-based testing. *Software Testing, Verification and Reliability*, 21(2):75–100, 2011.

- [20] Yih-Farn Chen, David S. Rosenblum, and Kiem-Phong Vo. TESTTUBE: a system for selective regression testing. In *Proceedings of the 16th International Conference on Software Engineering (ICSE-16)*, pages 211–220, May 1994.
- [21] Pavan Kumar Chittimalli and Mary Jean Harrold. Recomputing coverage information to assist regression testing. *IEEE Transactions on Software Engineering*, 35(4):452–469, 2009.
- [22] Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, Boston, 2001.
- [23] Myra B. Cohen, Matt B. Dwyer, and Jiangfan Shi. Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. *Software Engineering, IEEE Transactions on*, 34(5):633–650, 2008.
- [24] Myra B. Cohen, Matthew B. Dwyer, and Jiangfan Shi. Coverage and adequacy in software product line testing. In *Proceedings of the Workshop on Role of software architecture for testing and analysis (ROSATEA '06)*, pages 53–63, 2006.
- [25] Paulo Anselmo da Mota Silveira Neto, Ivan do Carmo Machado, John D. McGregor, Eduardo Santana de Almeida, and Silvio Romero de Lemos Meira. A systematic mapping study of software product lines testing. *Information and Software Technology*, 53(5):407–423, May 2011.
- [26] Paulo Anselmo da Mota Silveira Neto, Ivan do Carmo Machado, Yguaratã Cerqueira Cavalcanti, Eduardo Santana de Almeida, Vicinius Cardoso Garcia, and Silvio Romero de Lemos Meira. A regression testing approach for software product lines architectures. *Proceedings of the Fourth Brazilian Symposium on Software Components, Architectures and Reuse*, pages 41–50, 2010.
- [27] Paulo Anselmo da Mota Silveira Neto, Per Runeson, Ivan do Carmo Machado, Eduardo Santana de Almeida, Silvio Romero de Lemos Meira, and Emelie Engström. Testing software product lines. *IEEE Software*, 28(5):16–20, October 2011.
- [28] Stephan Diehl. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Springer, 1 edition, May 2007.
- [29] Stephan Diehl, Fabian Beck, and Michael Burch. Uncovering strengths and weaknesses of radial visualizations—an empirical approach. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):935–942, December 2010.

-
- [30] Christian Dinnus and Klaus Pohl. Experiences with software product line engineering. In *Software Product Line Engineering*, pages 413–434. Springer-Verlag, Berlin/Heidelberg, 2005.
- [31] Tore Dybå. An empirical investigation of the key factors for success in software process improvement. *IEEE Transactions on Software Engineering*, 31(5):410–424, May 2005.
- [32] Tore Dybå, Barbara A. Kitchenham, and Magne Jørgensen. Evidence-based software engineering for practitioners. *IEEE Software*, 22(1):58–65, January 2005.
- [33] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. Selecting empirical methods for software engineering research. In *Guide to Advanced Empirical Software Engineering*, pages 285–311. Springer London, London, 2008.
- [34] Sebastian Elbaum, Alexey G. Malishevsky, and Gregg Rothermel. Prioritizing test cases for regression testing. *SIGSOFT Softw. Eng. Notes*, 25(5):102–112, August 2000.
- [35] Emelie Engström and Per Runeson. Decision support for test management and scope selection in a software product line context. In *Proceedings of the IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW '11)*, pages 262–265, 2011.
- [36] 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 (ICST'11)*, pages 367–376, 2011.
- [37] Emelie Engström, Per Runeson, and Mats Skoglund. A systematic review on regression test selection techniques. *Information and Software Technology*, 52(1):14–30, January 2010.
- [38] 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 (ICST'10)*, pages 75–78, 2010.
- [39] Yankui Feng, Xiaodong Liu, and Jon Kerridge. A product line based aspect-oriented generative unit testing approach to building quality components. In *Proceedings of the 31st Annual International Computer Software and Applications Conference*, pages 403–408, July 2007.

- [40] Kurt F. Fischer. A test case selection method for the validation of software maintenance modifications. In *Proceedings of the International Computer Software and Applications Conference*, pages 421–426, 1977.
- [41] Birgit Geppert, Jenny Li, Frank Röbler, and David M. Weiss. Towards generating acceptance tests for product lines. In *Software Reuse: Methods, Techniques, and Tools*, volume 3107 of *Lecture Notes in Computer Science*, pages 35–48. Springer Berlin Heidelberg, 2004.
- [42] Tony Gorschek, Claes Wohlin, Per Carre, and Stig Larsson. A model for technology transfer in practice. *IEEE Software*, 23(6):88–95, December 2006.
- [43] Denis Gračanin, Krešimir Matković, and Mohamed Eltoweissy. Software visualization. *Innovations in Systems and Software Engineering*, 1(2):221–230, 2005.
- [44] Mats Grindal, Birgitta Lindström, Jeff Offutt, and Sten Andler. An evaluation of combination strategies for test case selection. *Empirical Software Engineering*, 11(4):583–611, 2006. 10.1007/s10664-006-9024-2.
- [45] Jo Hannay, Dag Sjøberg, and Tore Dybå. A systematic review of theory use in software engineering experiments. *IEEE Transactions on Software Engineering*, 33(2):87–107, February 2007.
- [46] Michael Harder, Jeff Mellen, and Michael D. Ernst. Improving test suites via operational abstraction. In *Proceedings 25th International Conference on Software Engineering*, pages 60–71, 2003.
- [47] Mary Jean Harrold, Rajiv Gupta, and Mary Lou Soffa. A methodology for controlling the size of a test suite. *ACM Transactions on Software Engineering and Methodology*, 2(3):270–285, July 1993.
- [48] Jean Hartmann, Marlon Vieira, and Axel Ruder. A UML-based approach for validating product lines. In *Proceedings of the International Workshop on Software Product Line Testing (SPLiT 2004)*, pages 58–65, August 2004.
- [49] William A. Hetrick, Charles W. Krueger, and Joseph G. Moore. Incremental return on incremental investment: Engenio’s transition to software product line practice. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 798–804, 2006.
- [50] Alan Hevner. A three cycle view of design science research. *Scandinavian Journal of Information Systems*, 19(2), January 2007.
- [51] Alan Hevner and Samir Chatterjee. *Design Research in Information Systems: Theory and Practice*. Springer, May 2010.

-
- [52] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75–105, 2004.
- [53] Joseph R. Horgan and Saul London. A data flow coverage testing tool for c. In *Proceedings of the Second Symposium on Assessment of Quality Software Development Tools*, pages 2–10, 1992.
- [54] John Hughes, Val King, Tom Rodden, and Hans Andersen. Moving out from the control room: ethnography in system design. In *Proceedings of the ACM conference on Computer supported cooperative work (CSCW '94)*, pages 429–439, 1994.
- [55] IEEE. Standard glossary of software engineering terminology. Technical Report 610.12-1990, 1990.
- [56] IEEE. Standard for software test documentation. Technical Report 829-1983, Revision, 1998.
- [57] Martin Ivarsson and Tony Gorschek. A method for evaluating rigor and industrial relevance of technology evaluations. *Empirical Software Engineering*, 16(3):365–395, 2011.
- [58] Michel Jaring, René L. Krikhaar, and Jan Bosch. Modeling variability and testability interaction in software product line engineering. In *Seventh International Conference on Composition-Based Software Systems (ICCBSS)*, pages 120–129, 2008.
- [59] Martin Fagereng Johansen, Oystein Haugen, and Frank Fleurey. A survey of empirics of strategies for software product line testing. In *Proceedings of the IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 266–269, March 2011.
- [60] James A. Jones, Mary Jean Harrold, and John Stasko. Visualization of test information to assist fault localization. In *Proceedings of the ACM 24th International Conference on Software Engineering (ICSE '02)*, pages 467–477, 2002.
- [61] Natalia Juristo, Ana Moreno, Sira Vegas, and Martin Solari. In search of what we experimentally know about unit testing. *IEEE Software*, 23(6):72–80, November 2006.
- [62] Jussi Kasurinen, Ossi Taipale, and Kari Smolander. How test organizations adopt new testing practices and methods? In *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 553–558, March 2011.

- [63] Shiego Kato and Nobuhito Yamaguchi. Variation management for software product lines with cumulative coverage of feature interactions. In *Proceedings of the 15:th international Software Product Line Conference*, pages 140–149, Munich, Germany, 2011. IEEE Computer Society.
- [64] Holger M. Kienle and Hausi A. Müller. Requirements of software visualization tools: A literature survey. In *IEEE Proceedings of the 4th International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2007)*, pages 2–9, June 2007.
- [65] Chang Hwan Peter Kim, Sarfraz Khurshid, and Don Batory. Shared execution for efficiently testing product lines. In *Proceedings of the IEEE International Symposium on Software Reliability Engineering (ISSRE 2012)*, November 2012.
- [66] Jung-Min Kim and Adam Porter. A history-based test prioritization technique for regression testing in resource constrained environments. In *Proceedings of the ACM 24th International Conference on Software Engineering*, pages 119–129, 2002.
- [67] Tomoji Kishi and Natsuko Noda. Design testing for product line development based on test scenarios. In *Proceedings of the International Workshop on Software Product Line Testing (SPLiT 2004)*, pages 19–26, August 2004.
- [68] Barbara A. Kitchenham, Shari Lawrence Pfleeger, Lesley M. Pickard, Peter W. Jones, David C. Hoaglin, Khaled El-Emam, and Jarrett Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering*, 28(8):721–734, August 2002.
- [69] Ronny Kolb. A risk driven approach for efficiently testing software product lines. In *5th GPCE Young, Researches Workshop*, Erfurt, Germany, 2003.
- [70] Ronny Kolb and Dirk Muthig. Making testing product lines more efficient by improving the testability of product line architectures. In *Proceedings of the ACM ISSA workshop on Role of software architecture for testing and analysis (ROSATEA '06)*, pages 22–27, 2006.
- [71] Bogdan Korel, Luay Ho Tahat, and Mark Harman. Test prioritization using system models. In *Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM'05)*, pages 559–568, 2005.
- [72] Bogdan Korel, Luay Ho Tahat, and Boris Vaysburg. Model based regression test reduction using dependence analysis. In *Proceedings of the 10th Annual Software Reliability Symposium International Conference on Software Maintenance*, pages 214–223, 2002.

-
- [73] Rainer Koschke. Software visualization for reverse engineering. In Stephan Diehl, editor, *Software Visualization*, number 2269 in Lecture Notes in Computer Science, pages 138–150. Springer Berlin Heidelberg, January 2002.
- [74] Craig Larman and Victor R. Basili. Iterative and incremental developments. a brief history. *Computer*, 36(6):47–56, June 2003.
- [75] Hareton K. N. Leung and Lee White. A study of integration testing and software regression at the integration level. In *Proceedings of the IEEE Conference on Software Maintenance*, pages 290–301, November 1990.
- [76] Hareton K. N. Leung and Lee White. A cost model to compare regression test strategies. In *Proceedings of the Conference on Software Maintenance*, pages 201–208, October 1991.
- [77] J. Jenny Li, Birgit Geppert, Frank Rößler, and David M. Weiss. Reuse execution traces to reduce testing of product lines. In *Proceedings of the 11th International Conference Software Product Lines. Second Volume (Workshops)*, pages 65–72, 2007.
- [78] J. Jenny Li, David M. Weiss, and J. Hamilton Slye. Automatic integration test generation from unit tests of eXVantage product family. In *Proceedings of the 11th International Conference Software Product Lines. Second Volume (Workshops)*, pages 73–80, 2007.
- [79] Frank J. van der Linden, Klaus Schmid, and Eelco Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer, 1 edition, July 2007.
- [80] Teng Long, Ilchul Yoon, Adam Porter, Alan Sussman, and Atif M. Memon. Overlap and synergy in testing software components across loosely-coupled communities. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA'12)*, 2012.
- [81] Jonathan I Maletic, Daniel J. Mosora, Christian D. Newman, Michael L. Collard, Andrew Sutton, and Brian P. Robinson. MosaiCode: visualizing large scale software: A tool demonstration. pages 1–4, September 2011.
- [82] John D. McGregor. Testing a software product line. Technical Report CMU/SEI-2001-TR-022, ESC-TR-2001-022, Software Engineering Institute, Carnegie Mellon University, 2001.
- [83] John D. McGregor. Testing a software product line. In *Testing Techniques in Software Engineering*, volume 6153 of *Lecture Notes in Computer Science*, pages 104–140. Springer Berlin / Heidelberg, 2010.

- [84] John D. McGregor and Kyungsoo Im. The implications of variation for testing in a software product line. In *Proceedings of the International Workshop on Software Product Line Testing (SPLiT 2007)*, 2007.
- [85] Satish Mishra. Specification based software product line testing: A case study. In *Concurrency, Specification and Programming (CS&P 2006)*, 2006.
- [86] Michela Montesi and Patricia Lago. Software engineering article types: An analysis of the literature. *Journal of Systems and Software*, 81(10):1694–1714, October 2008.
- [87] Michela Montesi and John Mackenzie Owen. Research journal articles as document genres: exploring their role in knowledge organization. *Journal of Documentation*, 64(1):143–167, January 2008.
- [88] Henry Muccini and André Van Der Hoek. Towards testing product line architectures. In *Proceedings of International Workshop on Testing and Analysis of Component Based Systems*, pages 111–121, 2003.
- [89] Clémentine Nebut, Franck Fleurey, Yves Le Traon, and Jean-marc Jézéquel. A requirement-based approach to test product families. *Proceedings of the 5th workshop on product families engineering*, pages 198–210, 2003.
- [90] Ralf Nörenberg, Anastasia Cmyrev, Ralf Reißing, and Klaus D. Müller-Glaser. An efficient specification-based regression test selection technique for E/E-Systems. In *Workshop Automotive Software Engineering*, 2011.
- [91] A. Jefferson Offutt, Jie Pan, and Jeffrey M. Voas. Procedures for reducing the size of coverage-based test sets. In *Proceedings of the 12th International Conference on Testing Computer Software*, pages 111–123, 1995.
- [92] Akira K. Onoma, Wei-Tek Tsai, Mustafa Poonawala, and Hiroshi Suganuma. Regression testing in an industrial environment. *Communications of the ACM*, 41(5):81–86, May 1998.
- [93] Alessandro Orso, Nanjuan Shi, and Mary Jean Harrold. Scaling regression testing to large software systems. In *Proceedings of the ACM SIGSOFT 12th International symposium on Foundations of software engineering (FSE-12)*, volume 29, pages 241–251, October 2004.
- [94] Sebastian Oster, Andreas Wübbecke, Gregor Engels, and Andy Schürr. A survey of model-based software product lines testing. In *Model-based Testing for Embedded Systems, Computational Analysis, Synthesis, and Design of Dynamic systems*, pages 339–381. CRC Press/Taylor & Francis, 2011.

- [95] Raja Parasuraman, Thomas B. Sheridan, and Christopher D. Wickens. A model for types and levels of human interaction with automation. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 30(3):286–297, May 2000.
- [96] Greg Paula. Reinventing a core product line. *Mechanical Engineering*, 119(10):102–103, 1997.
- [97] Kai Petersen and Claes Wohlin. Context in industrial software engineering research. In *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM '09)*, pages 401–404, October 2009.
- [98] Andreas Pleuss, Rick Rabiser, and Goetz Botterweck. Visualization techniques for application in interactive product configuration. In *Proceedings of the 15th International Software Product Line Conference*, volume 2 of *SPLC '11*, pages 22:1–22:8, 2011.
- [99] Klaus Pohl, Günther Böckle, and Frank J. Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag, 1 edition, September 2005.
- [100] Klaus Pohl and Andreas Metzger. Software product line testing. *Communications of the ACM*, 49:78, December 2006.
- [101] Colin Potts. Software-engineering research revisited. *IEEE Software*, 10(5):19–28, September 1993.
- [102] Xiao Qu, Myra B. Cohen, and Gregg Rothermel. Configuration-aware regression testing: an empirical study of sampling and prioritization. In *Proceedings of the 2008 international symposium on Software testing and analysis (ISSTA '08)*, pages 75–86, 2008.
- [103] Austen Rainer and Sarah Beecham. A follow-up empirical evaluation of evidence based software engineering by undergraduate students. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, 2008.
- [104] Austen Rainer, Tracy Hall, and Nathan Baddoo. A preliminary empirical investigation of the use of evidence based software engineering by undergraduate students. In *Proceedings of the 10th International Conference on Evaluation and Assessment in Software Engineering*, 2006.
- [105] Austen Rainer, Dorota Jagielska, and Tracy Hall. Software engineering practice versus evidence-based software engineering research. In *Proceedings of the ACM Workshop on Realising evidence-based software engineering (REBSE '05)*, pages 1–5, 2005.

-
- [106] Sacha Reis, Andreas Metzger, and Klaus Pohl. Integration testing in software product line engineering: a model-based technique. In *Proceedings of the 10th international conference on Fundamental approaches to software engineering*, pages 321–335, 2007.
- [107] Andreas Reuys, Erik Kamsties, Klaus Pohl, and Sacha Reis. Model-based system testing of software product families. In *Advanced Information Systems Engineering*, volume 3520, pages 519–534. Springer Berlin Heidelberg, 2005.
- [108] Andreas Reuys, Sacha Reis, Erik Kamsties, and Klaus Pohl. Derivation of domain test scenarios from activity diagrams. In *Proceedings of the International Workshop on Product Line Engineering: The Early Steps: Planning, Modeling, and Managing (PLEES'03)*, Erfurt, 2003.
- [109] Andreas Reuys, Sacha Reis, Erik Kamsties, and Klaus Pohl. The ScenTED method for testing software product lines. In *Proceedings of the Software Product Lines - Research Issues in Engineering and Management*, pages 479–520, 2006.
- [110] John Rooksby, Mark Rouncefield, and Ian Sommerville. Testing in the wild: The social and organisational dimensions of real world practice. *Computer Supported Cooperative Work (CSCW)*, 18(5):559–580, 2009.
- [111] Gregg Rothermel and Mary Jean Harrold. Analyzing regression test selection techniques. *IEEE Transactions on Software Engineering*, 22(8):529–551, August 1996.
- [112] Gregg Rothermel and Mary Jean Harrold. A safe, efficient regression test selection technique. *ACM Transactions on Software Engineering and Methodology*, 6(2):173–210, April 1997.
- [113] Gregg Rothermel, Mary Jean Harrold, and Jeinay Dedhia. Regression test selection for c++ software. *Software Testing, Verification and Reliability*, 10(2):77–109, June 2000.
- [114] Gregg Rothermel, Mary Jean Harrold, Jeffery Ostrin, and Christie Hong. An empirical study of the effects of minimization on the fault detection capabilities of test suites. In *Proceedings of the IEEE International Conference on Software Maintenance*, page 34, 1998.
- [115] Gregg Rothermel, Roland H. Untch, Chu Chengyun, and Mary Jean Harrold. Test case prioritization: an empirical study. In *Proceedings of the IEEE International Conference on Software Maintenance*, pages 179–188, 1999.

- [116] Per Runeson and Emelie Engström. Regression testing in software product line engineering. volume 86 of *Advances in Computers*, pages 223–263. Elsevier, 2012.
- [117] Per Runeson and Emelie Engström. Software product line testing—a 3D regression testing problem. In *Proceedings of the IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST)*, pages 742–746, April 2012.
- [118] Per Runeson, Martin Höst, Austen Rainer, and Björn Regnell. *Case Study Research in Software Engineering—Guidelines and Examples*. Wiley, 2012.
- [119] Per Runeson, Mats Skoglund, and Emelie Engström. Test benchmarks—what is the question? In *Proceedings of the IEEE International Conference on Software Testing Verification and Validation Workshop (ICSTW '08)*, pages 368–371, April 2008.
- [120] Giedre Sabaliauskaite, Annabella Loconsole, Emelie Engström, Michael Unterkalmsteiner, Björn Regnell, Per Runeson, Tony Gorschek, and Robert Feldt. Challenges in aligning requirements engineering and verification in a large-scale industrial context. In *Requirements Engineering: Foundation for Software Quality*, volume 6182 of *Lecture Notes in Computer Science*, pages 128–142. Springer Berlin / Heidelberg, 2010.
- [121] A. S. M. Sajeev and Bugi Wibowo. Regression test selection based on version changes of components. In *Proceedings of the IEEE Tenth Asia-Pacific Software Engineering Conference (APSEC '03)*, pages 78–85, 2003.
- [122] Mats Skoglund and Per Runeson. A case study of the class firewall regression test selection technique on a large scale distributed software system. In *International Symposium on Empirical Software Engineering*, pages 72–81, 2005.
- [123] Vanessa Stricker, Andreas Metzger, and Klaus Pohl. Avoiding redundant testing in application engineering. In *Proceedings of the 14th international conference on Software product lines: going beyond*, Lecture Notes in Computer Science, pages 226–240. Springer Berlin Heidelberg, 2010.
- [124] David P. Tegarden. Business information visualization. *Communications of the Association for Information Systems*, 1(4), January 1999.
- [125] Antti Tevanlinna, Juha Taina, and Raine Kauppinen. Product family testing: a survey. *SIGSOFT Software Engineering Notes*, 29(2):12, March 2004.
- [126] Paolo Tonella, Paolo Avesani, and Angelo Susi. Using the case-based ranking methodology for test case prioritization. In *Proceedings of the IEEE 22nd International Conference on Software Maintenance (ICSM '06)*, pages 123–133, 2006.

- [127] Tim Trew. What design policies must testers demand from product line architects? In *Proceedings of the International Workshop on Software Product Line Testing*, Technical Report: ALR-2004-031, pages 51–57. Avaya Labs, 2004.
- [128] Karl T. Ulrich and Steven D. Eppinger. *Product Design and Development*. McGraw-Hill, Inc., 1995.
- [129] Lee White and Khalil Abdullah. A firewall approach for the regression testing of object-oriented software. In *Proceedings of 10th annual software quality week*, 1997.
- [130] Lee White and Brian Robinson. Industrial real-time regression testing and analysis using firewalls. In *Proceedings of the 20th IEEE International Conference on Software Maintenance (ICSM '04)*, pages 18–27, 2004.
- [131] Bernd Wilhelm. Platform and modular concepts at volkswagen—their effects on the assembly process. In *Transforming Auto Assembly*, pages 146–156, Berlin, Germany, 1997. Springer-Verlag.
- [132] Claes Wohlin, Per Runeson, Paulo Anselmo da Mota Silveira Neto, Emelie Engström, Ivan do Carmo Machado, and Eduardo Santana de Almeida. On the reliability of mapping studies in software engineering. *Manuscript submitted for publication*, 2013.
- [133] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, 2000.
- [134] W. Eric Wong, Joseph R. Horgan, Saul London, and Hira Agrawal Bellcore. A study of effective regression testing in practice. In *Proceedings of the IEEE Eighth International Symposium on Software Reliability Engineering (ISSRE '97)*, pages 264–274, 1997.
- [135] Cemal Yilmaz, Myra B. Cohen, and Adam A. Porter. Covering arrays for efficient fault characterization in complex configuration spaces. *IEEE Transactions on Software Engineering*, 32(1):20–34, January 2006.
- [136] Robert K. Yin. *Case Study Research: Design and Methods*. SAGE, 2003.
- [137] Shin Yoo and Mark Harman. Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification and Reliability*, 22(2):67–120, March 2012.
- [138] Shin Yoo, Mark Harman, Paolo Tonella, and Angelo Susi. Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge. In *Proceedings of the ACM 18th International symposium on Software testing and analysis (ISSTA '09)*, pages 201–212, 2009.

- [139] Hui Zeng, Wendy Zhang, and David Rine. Analysis of testing effort by using core assets in software product line testing. In *Proceedings of the International Workshop on Software Product Line Testing*, Technical Report: ALR-2004-031, pages 1–6. Avaya Labs, 2004.
- [140] Jiang Zheng, Brian Robinson, Laurie Williams, and Karen Smiley. An initial study of a lightweight process for change identification and regression test selection when source code is not available. In *Proceedings of the IEEE 16th International Symposium on Software Reliability Engineering (ISSRE 2005)*, pages 225–234, November 2005.

INCLUDED PAPERS

A SYSTEMATIC REVIEW ON REGRESSION TEST SELECTION TECHNIQUES

Abstract

Regression testing is verifying that previously functioning software remains after a change. With the goal of finding a basis for further research in a joint industry-academia research project, we conducted a systematic review of empirical evaluations of regression test selection techniques. We identified 27 papers reporting 36 empirical studies, 21 experiments and 15 case studies. In total 28 techniques for regression test selection are evaluated. We present a qualitative analysis of the findings, an overview of techniques for regression test selection and related empirical evidence. No technique was found clearly superior since the results depend on many varying factors. We identified a need for empirical studies where concepts are evaluated rather than small variations in technical implementations.

Emelie Engström, Per Runeson and Mats Skoglund
Journal of Information and Software Technology 52(1):14-30, 2010

1 Introduction

Efficient regression testing is important, even crucial, for organizations with a large share of their cost in software development. It includes, among other tasks, determining which test cases need to be re-executed, i.e. regression test selection, in order to verify the behavior of modified software. Regression test selection involves a trade-off between the cost for re-executing test cases, and the risk for missing faults introduced through side effects of changes to the software. Iterative development strategies and reuse are common means of saving time and effort for the development. However they both require frequent retesting of previously

tested functions due to changes in related code. The need for efficient regression testing strategies is thus becoming more and more important.

A great deal of research effort has been spent on finding cost-efficient methods for different aspects of regression testing. Examples include test case selection based on code changes [1] [7] [14] [18] [21] [23] [24] [50] [64] [59] [67] and specification changes [40] [42] [55] [6], evaluation of selection techniques [49], change impact analysis [45], regression tests for different applications e.g. database applications [19], regression testing of GUIs and test automation [41], and test process enhancement [33]. To bring structure to the topics, researchers have typically divided the field of regression testing into i) test selection, ii) modification identification, iii) test execution, and iv) test suite maintenance. This review is focused on test selection techniques for regression testing.

Although techniques for regression test selection have been evaluated in previous work [3] [16] [38] [60], no general solution has been put forward since no technique could possibly respond adequately to the complexity of the problem and the great diversity in requirements and preconditions in software systems and development organizations. Neither does any single study evaluate every aspect of the problem; e.g. Kim et al. [29] evaluate the effects of regression test application frequency, Elbaum et al. [12] investigate the impact that different modifications have on regression test selection techniques, several studies examine the ability to reduce regression testing effort [3] [12] [16] [29] [38] [60] [66] and to reveal faults [12] [16] [29] [50].

In order to map the existing knowledge in the field, we launched a systematic review to collect and compare existing empirical evidence on regression test selection. The use of systematic reviews in the software engineering domain has been subject to a growing interest in the last years. In 2004 Kitchenham proposed a guideline adapted to the specific characteristics of software engineering research. This guideline has been followed and evaluated [5] [30] [58] and updated accordingly in 2007 [31]. Kitchenham et al. recently published a review of 20 systematic reviews in software engineering 2004-2007 [32]. Ideally, several empirical studies identified in a systematic review evaluate the same set of techniques under similar conditions on different subject programs. Then there would be a possibility to perform an aggregation of findings or even meta-analysis and thus enable drawing general conclusions. However, as the field of empirical software engineering is quite immature, systematic reviews have not given very clear pictures of the results. In this review we found that the existing studies were diverse, thus hindering proper quantitative aggregation. Instead we present a qualitative analysis of the findings, an overview of existing techniques for regression test selection and of the amount and quality of empirical evidence. There are surveys and reviews of software testing research published before, but none of these has the broad scope and the extensive approach of a systematic review. In 2004 Do et al. presented a survey of empirical studies in software testing in general [9] including regression testing. Their study covered two journals and four confer-

ences over ten years (1994-2003). Other reviews of regression test selection are not exhaustive but compare a limited number of chosen regression test selection techniques. Rothermel and Harrold presented a framework for evaluating regression test techniques already in 1996 [49] and evaluated the, by that time, existing techniques. Juristo et al. aggregated results from unit testing experiments [27] of which some evaluate regression testing techniques, although with a more narrow scope. Binkley et al. reviewed research on the application of program slicing to the problem of regression testing [4]. Hartman et al. reports a survey and critical assessment of regression testing tools [22]. However, as far as we know, no systematic review on regression test selection research has been carried through since the one in 1996 [49]. An early report of this study was published in 2008 [13], which here is further advanced especially with respect to the detailed description of the techniques (Section 3.4), their development history and the analysis of the primary studies (Section 3.5)¹.

This paper is organized as follows. In section 2 the research method used for our study is described. Section 3 reports the empirical studies and our analyses. Section 4 discusses the results and section 5 concludes the work.

2 Research Method

2.1 Research Questions

This systematic review aims at summarizing the current state of the art in regression test selection research by proposing answers to a set of questions below. The research questions stem from a joint industry-academia research project, which aims at finding efficient procedures for regression testing in practice. We searched for candidate regression test selection techniques that were empirically evaluated, and in case of lack of such techniques, to identify needs for future research. Further, as the focus is on industrial use, issues of scale-up to real-size projects and products are important in our review. The questions are:

- RQ1) Which techniques for regression test selection in the literature have been evaluated empirically?
- RQ2) Can these techniques be classified, and if so, how?
- RQ3) Are there significant differences between these techniques that can be established using empirical evidence?
- RQ4) Can technique A be shown to be superior to technique B, based on empirical evidence?

¹In this extended analysis, some techniques that originally were considered different ones, were considered the same technique. Hence, the number of techniques differ from [10]. Further, the quality of two empirical studies was found insufficient in the advanced analysis, why two studies were removed.

Answers to these research questions are searched in the published literature using the procedures of systematic literature reviews as proposed by Kitchenham [31].

2.2 Sources of information

In order to gain a broad perspective, as recommended in Kitchenham's guidelines [31], we searched widely in electronic sources. The advantage of searching databases rather than a limited set of journals and conference proceedings, is also empirically motivated by Dieste et al [8]. The following seven databases were covered:

- Inspec (<www.theiet.org/publishing/inspec/>)
- Compendex (<www.engineeringvillage2.org/>)
- ACM Digital Library (<portal.acm.org/>)
- IEEE eXplore (<ieeexplore.ieee.org/>)
- ScienceDirect (<www.sciencedirect.com/>)
- Springer LNCS (<www.springer.com/lncs/>)
- Web of Science (<www.isiknowledge.com/>)

These databases cover the most relevant journals and conference and workshop proceedings within software engineering, as confirmed by Dybå et al. [11]. Grey literature (technical reports, some workshop reports, work in progress) was excluded from the analysis for two reasons: the quality of the grey literature is more difficult to assess and the volume of studies included in the first searches would have grown unreasonably. The searches in the sources selected resulted in overlap among the papers, where the duplicates were excluded primarily by manual filtering.

2.3 Search criteria

The initial search criteria were broad in order to include articles with different uses of terminology. The key words used were <regression> and (<test> or <testing>) and <software>, and the database fields of title and abstract were searched. The start year was set to 1969 to ensure that all relevant research within the field would be included, and the last date for inclusion is publications within 2006. The earliest primary study actually included was published in 1997. Kitchenham recommends that exclusion based on languages should be avoided [31]. However, only papers written in English are included. The initial search located 2 923 potentially relevant papers.

2.4 Study Selection

In order to obtain independent assessments, four researchers were involved in a three-stage selection process, as depicted in Figure 1. In the first stage duplicates and irrelevant papers were excluded manually based on titles. In our case, the share of irrelevant papers was extremely large since papers on software for statistical regression testing or other regression testing could not be distinguished from papers on software regression testing in the database search. The term software did not distinguish between the two areas, since researchers on statistical regression testing often develop some software for their regression test procedures. After the first stage 450 papers remained.

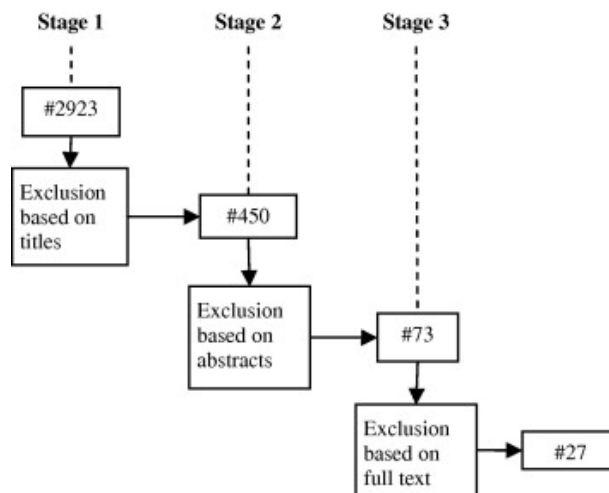


Figure 1: Study selection procedure.

In the second stage, information in abstracts was analyzed and the papers were classified along two dimensions: research approach and regression testing approach. Research approaches were experiment, case study, survey, review, theory and simulation. The two latter types were excluded, as they are not presenting an empirical research approach, and the survey and review papers were not considered as being primary studies but rather related work to the systematic review. At this stage we did not judge the quality of the empirical data. Regression testing approaches were selection, reduction, prioritization, generation, execution and other. Only papers focusing on regression test selection were included.

In the third stage a full text analysis was performed on the 73 papers and the empirical quality of the studies was further assessed. The following questions were asked in order to form quality criteria for which studies to exclude before the final data extraction:

- Is the study focused on a specific regression test selection method? E.g. a paper could be excluded that presents a method that potentially could be used for regression testing, but is evaluated from another point of view.
- Are the metrics collected and the results relevant for a comparison of methods? E.g. a paper could be excluded which only reports on the ability to predict fault prone parts of the code, but not on the fault detection effectiveness or the cost of the regression test selection strategy.
- Is data collected and analyzed in a sufficiently rigorous manner? E.g. a paper could be excluded if a subset of components was analyzed and conclusions were drawn based on those, without any motivation for the selection.

These questions are derived from a list of questions, used for a similar purpose, published by Dybå et al. [11]. However in our review context, quality requirements for inclusion had to be weaker than suggested by Dybå et al. in order to obtain a useful set of studies to compare. The selection strategy was in general more inclusive than exclusive. Only papers with very poorly reported or poorly conducted studies were excluded, as well as papers where the comparisons made were considered irrelevant to the original goals of this study.

Abstract analysis and full text analysis were performed in a slightly iterative fashion. Firstly, the articles were independently assessed by two of the researchers. In case of disagreement, the third researcher acted as a checker. In many cases, disagreement was due to insufficient specification of the criteria. Hence, the criteria were refined and the analysis was continued.

In order to get a measure of agreement in the study selection procedure, the Kappa coefficient was calculated for the second stage, which comprised most judgments in the selection. In the second stage 450 abstracts were assessed by two researchers independently. In 41 cases conflicting assessments were made which corresponds to the Kappa coefficient $K = 0.78$. According to Landis and Koch [35] this translates to a substantial strength of agreement.

2.5 Data extraction and synthesis

Using the procedure, described in the previous section, 27 articles were finally selected that reported on 36 unique empirical studies, evaluating 28 different techniques. The definition of what constitutes a single empirical study, and what constitutes a unique technique is not always clear cut. The following definitions have been used in our study:

- Study: an empirical study applying a technique to one or more programs. Decisions on whether to split studies with multiple artifacts into different studies were based on the authors' own classification of the primary studies. Mostly, papers including studies on both small and large programs are presented as two different studies.

- **Technique:** An empirically evaluated method for regression test selection. If the only difference between two methods is an adaption to a specific programming language (e.g. from c++ to java) they are considered being the same technique.

Studies were classified according to type and size, see Section 2.1. Two types of studies are included in our review, experiments and case studies. We use the following definitions:

- **Experiment:** A study in which an intervention is deliberately introduced to observe its effects [56].
- **Case study:** An empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between the phenomenon and context are not clearly evident [68].

Surveys and literature reviews were also considered in the systematic review, e.g. [49] and [27], but rather as reference point for inclusion of primary studies than as primary studies as such.

Regarding size, the studies are classified as small, medium or large (S, M, L) depending on the study artifact sizes. A small study artifact has less than 2,000 lines of code (LOC), a large study artifact has more than 100,000 LOC, and a medium sized study artifact is in between. The class limits are somewhat arbitrarily defined. In most of the articles the lines of code metric is clearly reported and thus this is our main measurement of size. But in some articles sizes are reported in terms of number of methods or modules, reported as the authors' own statement about the size or not reported at all.

The classification of the techniques is part of answering RQ2 and is further elaborated in Section 2.4.

2.6 Qualitative assessment of empirical results

The results from the different studies were qualitatively analyzed in categories of four key metrics: reduction of cost for test execution, cost for test case selection, total cost, and fault detection effectiveness, see Section 3.5. The "weight" of an empirical study was classified according to the scheme in Table 1. A study with more "weight" is considered contributing more to the overall conclusions. A unit of analysis in an experiment is mostly a version of a piece of code, while in a case study; it is mostly a version of a whole system or sub-system.

The results from the different studies were then divided into six different categories according to the classification scheme in Table 2. The classification is based on the study "weight" and the size of the difference in a comparative empirical study. As the effect sizes were rarely reported in the studies, the sizes of the differences are also qualitatively assessed. The categorization of results was

Table 1: “Weight” of empirical study.

Type and size of study	Light empirical study “weight”	Medium empirical study “weight”
Experiment (small) Case study (small-medium)	Analysis units < 10	Analysis units >= 10
Experiment (medium) Case study (large)	Analysis units < 4	Analysis units >= 4

Table 2: Classification scheme for qualitative assessment of the weight of empirical results.

	No difference	Difference of small size	Difference of large size
Medium empirical study “weight”	<i>Strong</i> indication of equivalence between the two compared techniques	<i>Weak</i> indication that one technique is superior to the other	<i>Strong</i> indication that one technique is superior to the other
Light empirical study “weight”	<i>Weak</i> indication of equivalence between the two compared techniques	<i>No</i> indication of differences or similarities	<i>Weak</i> indication that one technique is superior to the other

made by two researchers in parallel and uncertainties were resolved in discussions. Results are presented in Figures 5 - 8 in Section 2.5.

No difference Difference of small size Difference of large size Medium empirical study “weight” Strong indication of equivalence between the two compared techniques Weak indication that one technique is superior to the other Strong indication that one technique is superior to the other Light empirical study “weight” Weak indication of equivalence between the two compared techniques No indication of differences or similarities Weak indication that one technique is superior to the other

2.7 Threats to validity

Threats to the validity of the systematic review are analyzed according to the following taxonomy; construct validity, reliability, internal validity and external validity.

Construct validity reflects to what extent the phenomenon under study really represents what the researchers have in mind and what is investigated according to the research questions. The main threat here is related to terminology. Since the systematic review is based on a hierarchical structure of terms - regression test/testing consists of the activities modification identification, test selection, test execution and test suite maintenance - we might miss other relevant studies on test selection that are not specifically aimed for regression testing. However, this is a consciously decided limitation, which has to be taken into account in the use of

the results. Another aspect of the construct validity is assurance that we actually find all papers on the selected topic. We analyzed the list of publication fora and the list of authors of the primary studies to validate that no major forum or author was missed.

Reliability focuses on whether the data is collected and the analysis is conducted in a way that it can be repeated by other researchers with the same results. We defined a study protocol setting up the overall research questions, the overall structure of the study as well as initial definitions of criteria for inclusions/exclusion, classification and quality. The criteria were refined during the study based on the identification of ambiguity that could mislead the researchers.

In a systematic review, the decision process for inclusion and exclusion of primary studies is the major focus when it comes to reliability, especially in this case where another domain (statistics) also uses the term regression testing. Our countermeasures taken to reduce the reliability threat were to set up criteria and to use two researchers to classify papers in stages 2 and 3. In cases of disagreement, a third opinion is used. However, the Kappa analysis indicates strong agreements. One of the primary researchers was changed between stages 2 and 3. Still, the uncertainties in the classifications are prevalent and a major threat to reliability, especially since the quality standards for empirical studies in software engineering are not high enough. Research databases is another threat to reliability [11]. The threat is reduced by using multiple databases; still the non-determinism of some database searches is a major threat to the reliability of any systematic review.

Internal validity is concerned with the analysis of the data. Since no statistical analysis was possible due to the inconsistencies between studies, the analysis is mostly qualitative. Hence we link the conclusions as clearly as possible to the studies, which underpin our discussions.

External validity is about generalizations of the findings derived from the primary studies. Most studies are conducted on small programs and hence generalizing them to a full industry context is not possible. In the few cases where experiments are conducted in the small as well as case studies in the large, the external validity is reasonable, although there is room for substantial improvements.

3 Results

3.1 Primary studies

The goal of this study was to find regression test selection techniques that are empirically evaluated. The papers were initially obtained in a broad search in seven databases covering relevant journals, conference and workshop proceedings within software engineering. Then an extensive systematic selection process was carried out to identify papers describing empirical evaluations of regression test selection techniques. The results presented here thus give a good picture of the existing evidence base.

Out of 2 923 titles initially screened, 27 papers (P1-P27) on empirical evaluations of techniques for regression test selection remained until the final stage. These 27 papers report on 36 unique studies (S1-S36), see Table 3, and compare in total 28 different techniques for regression test selection for evaluation (T1-T28), see listing in Table 3.3 below, which constitutes the primary studies of this systematic review. Five reference techniques are also identified (REF1-REF5), e.g. re-test all (all test cases are selected) and random(25) (25% of the test cases are randomly selected). In case the studies are reported partially or fully in different papers, we generally refer to the most recent one as this contains the most updated study. When referring to the techniques, we do on the contrary refer to the oldest, considering it being the original presentation of the technique.

Table 3: Primary studies, S1-S36, published in papers P1-P27, evaluation techniques T1-T28.

Study ID	Publication ID	Reference	Techniques	Artifacts	Type of study	Size of study
S1	P1	Baradhi and Mansour (1997) [2]	T4, T5, T6, T11, T12	Own unspecified	Exp	S
S2	P2	Bible et al. (2001) [3]	REF1 T7, T8	7x Siemens, Small constructed programs, constructed, realistic non-coverage based test suites	Exp	S
S3	P2	Bible et al. (2001) [3]	REF1 T7, T8	Space, Real application, real faults, constructed test cases	Exp	S
S4	P2	Bible et al. (2001) [3]	REF1 T7, T8	Player, One module of a large software system constructed realistic test suites	Exp	M
S5	P3	Elbaum et al. (2003) [12]	REF1 T2, T4, T18	Bash, Grep, Flex and Gzip, Real, non-trivial C program, constructed test suites	CS (Mult)	M
S6	P4	Frankl et al. (2003) [15]	REF1 T7, T10	7xSiemens, Small constructed programs, constructed, realistic, non-coverage based test suites	Exp	S
S7	P5	Graves et al. (2001) [16]	REF1 T1, T2, T7	7xSiemens, Small constructed programs, constructed, realistic non-coverage based test suites; space, Real application, real faults, constructed test cases; player, One module of a large software system constructed realistic test suites	Exp	S M
S8	P6	Harrold et al. (2001) [20]	REF1, REF2, REF3, REF4 T15	Siena, Jedit, JMeter, RegExp, Real programs, constructed faults	Exp	S

continued on next page ...

Table 3 – continued from previous page

Study ID	Publication ID	Reference	Techniques	Artifacts	Type of study	Size of study
S9	P7	Kim et al. (2005) [29]	REF1 T2, T7, T8	7xSiemens, Small constructed programs, constructed, realistic non-coverage based test suites; Space, Real application, real faults, constructed test cases	Exp	S
S10	P8	Koju et al. (2003) [34]	REF1, REF2, REF3, REF4 T15	Classes in .net framework, Open source, real test cases	Exp	S
S11	P9	Mansour et al. (2001) [38]	REF1 T4, T5, T6, T12	20 small sized Modules	Exp	S
S12	P10	Mao and Lu (2005) [40]	T16, T17, T24	Triangle, eBookShop, ShipDemo, Small Constructed programs	CS	S
S13	P11	Orso et al. (2004) [43]	REF1 T9, T15, T19	Jaba, Daikon, JBoss, Real-life programs, original test suites	Exp	M L
S14	P12	Pasala and Bhowmick (2005) [44]	REF1 T20	Internet Explorer (client), IIS (web server), application (app. Server), An existing browser based system, real test cases	CS	NR
S15	P13	Rothermel and Harrold (1997) [50]	REF1 T7	7xSiemens, Small constructed programs, constructed, realistic non-coverage based test suites	Exp	S
S16	P13	Rothermel and Harrold (1997) [50]	REF1 T7	Player, One module of a large software system constructed realistic test suites	Exp	M
S17	P14	Rothermel and Harrold (1998) [52]	REF1 T7	7xSiemens, Small constructed programs, constructed, realistic non-coverage based test suites	Exp	S
S18	P14	Rothermel and Harrold (1998) [52]	REF1 T7	7xSiemens, Small constructed programs, constructed, realistic non-coverage based test suites	Exp	S
S19	P14	Rothermel and Harrold (1998) [52]	REF1 T7	7xSiemens, Small constructed programs, constructed, realistic non-coverage based test suites;	Exp	S
S20	P14	Rothermel and Harrold (1998) [52]	REF1 T7	Player, One module of a large software system constructed realistic test suites	Exp	M
			REF1			

continued on next page ...

Table 3 – continued from previous page

Study ID	Publication ID	Reference	Techniques	Artifacts	Type of study	Size of study
S21	P14	Rothermel and Harrold (1998) [52]	T7	Commerercial, Real application, real test suite	Exp	S
S22	P15	Rothermel et al. (2002) [46]	REF1 T8, T18	Emp-server, Open-source, server component, constructed test cases; Bash Open-source, real and constructed test cases	Exp	M
S23	P16	Rothermel et al. (2004) [47]	REF1 T2, T8, T18	Bash, Open-source, real and constructed test cases	Exp	M
S24	P16	Rothermel et al. (2004) [47]	REF1 T2, T8, T18	Emp-server, Open-source, server component, constructed test cases	Exp	M
S25	P17	Skoglund and Runeson (2005) [57]	REF1 T9, T21	Swedbank, Real, large scale, distributed, component-based, J2EE system, constructed, scenario-based test cases	CS	L
S26	P18	Vokolos and Frankl (1998) [60]	REF1 T10	ORACOLO2, Real industrial sub-systems, real modifications, constructed test cases	CS	M
S27	P19	White and Robinson (2004) [63]	REF1 T3	14 real ABB projects, Industrial, Real-time system	CS	L
S28	P19	White and Robinson (2004) [63]	REF5 T9	2 real ABB projects, Industrial, Real-time system	CS	L
S29	P20	White et al. (2005) [62]	REF5 T3, T9, T25	OO-telecommunication software system	CS	S
S30	P20	White et al. (2005) [62]	T3, T9, T25	OO - real-time software system	CS	L
S31	P21	Willmor and Embury (2005) [65]	T7, T22, T23	Compiere, James, Mp3cd browser, Open source systems, real modifications	CS	NR
S32	P22	Wong et al. (1997) [66]	REF1 T13	Space, Real application, real faults, constructed test cases	CS	S
S33	P23	Wu et al. (1999) [67]	REF1 T14	ATM-simulator, small constructed program	CS	S
			REF1			

continued on next page ...

Table 3 – continued from previous page

Study ID	Publication ID	Reference	Techniques	Artifacts	Type of study	Size of study
S34	P23	Wu et al. (1999) [67]	T14	Subsystem of a fully networked supervisory control and data analysis system	CS	M
S35	P24, P25, P26	Zheng et al. (2005) [70], Zheng et al. (2006) [71] Zheng (2005) [69]	REF1 T26, T28	ABB-internal, Real C/C++ application	CS	M
S36	P27, P25	Zheng et al. (2006) [72], Zheng et al. (2006) [71]	REF1 T27, T28	ABB-internal, Real C/C++ application	CS	M
			REF1			

In most of the studies, the analyses are based on descriptive statistics. Tabulated data or bar charts are used as a basis for the conclusions. In two studies (S23 and S24), published in the same paper (P16) [47] statistical analysis is conducted, using ANOVA.

3.2 Analyses of the primary studies

In order to explore the progress of the research field, and to validate that the selected primary studies reasonably cover our general expectations of which fora and which authors should be represented, we analyze, as an extension to RQ1, aspects of the primary studies as such: where they are published, who published them, and when. As defined in Section 1.5, a paper may report on multiple studies, and in some cases the same study is reported in more than one paper. Different researchers have different criteria for what constitutes a study. We have tried to apply a consistent definition of what constitutes a study. This distribution of studies over papers is shown in Table 4. Most papers (18 out of 27) report a single study, while few papers report more than one. Two papers report new analyses of earlier published studies. Note that many of the techniques are originally presented in papers without empirical evaluation, hence these papers are not included as primary studies in the systematic review, but referenced in Section 2.3 as sources of information about the techniques as such (Table 3.3).

The number of identified techniques in the primary studies is relatively high compared to the number of studies, 28 techniques were evaluated in 36 studies. In Table 5, the distribution of techniques over different studies is presented. One technique was present in 14 different studies, another technique in 8 studies etc.

Table 4: Distribution of number of papers after the number of studies each paper reports.

# reported studies in each paper	# papers	# studies
0 (re-analysis of another study)	2	0
1	18	18
2	5	10
3	1	3
5	1	5
Total	27	36

Table 5: Distribution of techniques after occurrences in number of studies

Represented number of studies	in	Number of techniques
14		1
8		1
5		2
4		1
3		2
2		7
1		14
Total		28

14 techniques only appear in one study, which is not satisfactory when trying to aggregate information from empirical evaluations of the techniques.

Table 6 lists the different publication fora in which the articles have been published. It is worth noting regarding the publication fora, that the empirical regression testing papers are published in a wide variety of journals and conference proceedings. Limiting the search to fewer journals and proceedings would have missed many papers, see Table 6.

The major software engineering journals and conferences are represented among the fora. It is not surprising that a conference on software maintenance is on the top, but we found, during the validity analysis, that the International Symposium on Software Testing and Analysis is not on the list at all. We checked the proceedings specifically and have also noticed that, for testing in general, empirical studies have been published there, as reported by Do et al. [9], but apparently not on regression test selection during the studied time period.

Table 7 lists authors with more than one publication. In addition to these 17 authors, five researchers have authored or co-authored one paper each. In the top of the author's list, we find the names of the most prolific researchers in the field of

Table 6: Number of papers in different publication fora

Publication Fora	Type	#	%
International Conference on Software Maintenance	Conference	5	18.5
ACM Transactions of Software Engineering and Methodology	Journal	3	11.1
International Symposium on Software Reliability Engineering	Conference	3	11.1
International Conference on Software Engineering	Conference	3	11.1
Asia-Pacific Software Engineering Conference	Conference	2	7.4
International Symposium on Empirical Software Engineering	Conference	2	7.4
IEEE Transactions of Software Engineering	Journal	1	3.7
Journal of Systems and Software	Journal	1	3.7
Software Testing Verification and Reliability	Journal	1	3.7
Journal of Software Maintenance and Evolution	Journal	1	3.7
ACM SIGSOFT Symposium on Foundations of SE	Conference	1	3.7
Automated Software Engineering	Conference	1	3.7
Australian SE Conference	Conference	1	3.7
International Conf on COTS-based Software Systems	Conference	1	3.7
Int. Conference on Object-Oriented Programming, Systems, Languages, and Applications	Conference	1	3.7
Total		27	100

regression test selection (Rothermel and Harrold). It is interesting to notice from the point of view of conducting empirical software engineering research, that there are two authors on the top list with industry affiliation (Robinson and Smiley).

The regression test selection techniques have been published from 1988 to 2006, as shown in Figure 2 and Table 3.3. The first empirical evaluations were published in 1997 (one case study and three experiments), hence the empirical evaluations have entered the scene relatively late. 12 out of the 28 techniques have been originally presented and evaluated in the same paper: T12-S11 and T13-S32 (1997); T14-S33-S34 (1999); T18-S5 (2003); T19-S13 (2004); T20-S14; T21-S25; T23-S31; T25-S29-S30 and T26-S35 (2005); T27-S33 and T28-S35 (2006).

Table 7: Researchers and number of publications

Name	#	Name	#
Rothermel G.	9	Baradhi G.	2
Harrold M. J.	5	Frankl P. G.	2
Robinson B.	5	Kim J. M.	2
Zheng J.	4	Mansour N.	2
Elbaum S. G.	3	Orso A.	2
Kallakuri P.	3	Porter A.	2
Malishevsky A.	3	White L.	2
Smiley K.	3	Vokolos F.	2
Williams L.	3		

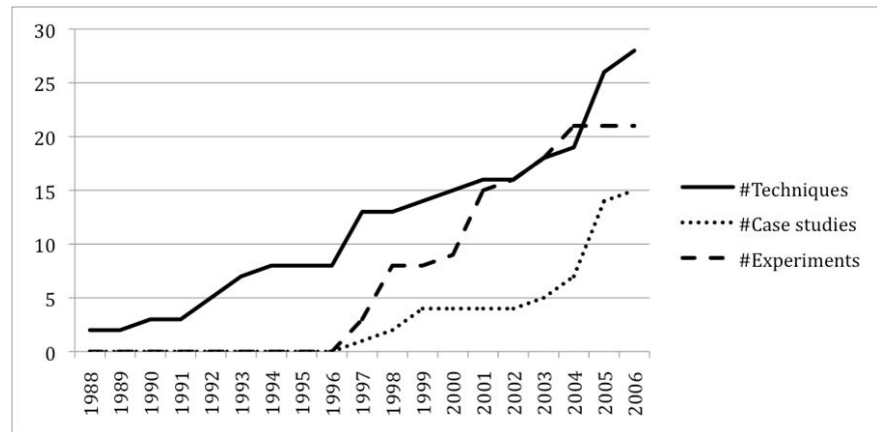


Figure 2: Accumulated number of published techniques, case studies and experiments.

We conclude from this analysis that there are only a few studies comparing many techniques in the same study, making it hard to find empirical data for a comprehensive comparison. However, some small and medium-sized artifacts have appeared as a de-facto benchmark in the field [9], enabling comparison to some extent of some techniques.

Most of the expected publication fora are represented, and one that is not represented, but was expected, was specifically double checked. Similarly, well known researchers in the field were among the authors, hence we consider the selected primary studies as being a valid set. It is clear from the publication analysis that the techniques published during the later years are published with empirical evaluations to a higher degree than during the earlier years, which is a positive trend in searching for empirically evaluated techniques as defined in RQ1.

3.3 Empirically evaluated techniques (RQ1)

Overview

Table 3.3 lists the 28 different regression test selection techniques (T1-T28), in chronological order according to date of first publication. In case the studies are reported partially or fully in different papers, we generally refer to the original one. In case a later publication has added details that are needed for exact specification of the technique, both references are used.

This list is specifically the answer to the first research question: which techniques for regression test selection existing in the literature have been evaluated empirically (RQ1). In this review, the techniques, their origin and description, are identified in accordance to what is stated in each of the selected papers, although

adapted according to our definition of what constitutes a unique technique in Section 1.5.

Development history

The historical development chain gives some advice on which techniques are related and how they are developed, see Figure 3. There are three major paths, beginning with T3, T7 and T8 respectively.

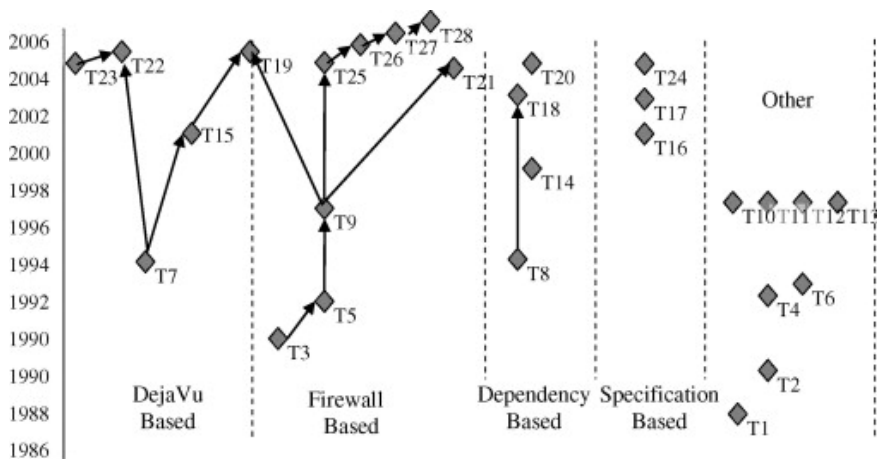


Figure 3: Evolution of techniques.

One group of techniques is the *firewall* techniques where dependencies to modified software parts are isolated inside a firewall. Test cases covering the parts within the firewall are selected for re-execution. The first firewall technique (T3) for procedural languages was presented by Leung and White in 1990 [37]. An empirical evaluation used a changed version (T5). The technique was adapted to object-oriented languages T9 in two similar ways [24] [61] and further enhanced and extended in the I-BACCI technique (T25-T28). It has also been adapted to Java (T21).

Another group of techniques is based on a technique invented by Rothermel and Harrold for procedural languages in 1993 [48] (T7), sometimes referred to as *DejaVu*. This technique has later been adopted to object-oriented languages T15 (for C++ [51], and for Java [20] [34]) and also further extended for MSIL code [34]. Through technique T19 it has also been combined with techniques from the group of firewall techniques. Extended techniques that cope with database state have also been created T22 and T23 [65].

The firewall techniques are based on relationships to changed software parts. Different granularities of parts have been used, such as dependencies between

Table 8: Techniques for regression test selection

Technique	Origin	Description	Evaluated in study
T1	Harrold and Soffa (1988) [21]	Dataflow-coverage-based	S7
T2	Fischer et al. (1981) [14] Hartman and Robson (1988) [23]	Modification-focused, minimization, branch and bound algorithm	S5, S7, S9, S23, S24
T3	Leung and White (1990) [37]	Procedural-design firewall	S27, S29, S30
T4	Gupta et al. (1992) [17]	Coverage-focused, slicing	S1, S5, S11
T5	White and Leung (1992) [64]	Firewall	S1, S11
T6	Agraval et al. (1993) [1]	Incremental	S1, S11
T7	Rothermel and Harrold (1993) [48]	Viewing statements, DejaVu	S2 -S4, S6, S7, S9, S15 - S21, S31
T8	Chen and Rosenblum (1994) [7]	Modified entity - TestTube	S2 - S4, S9, S22 - 24
T9	Pei et al. (1997) [24] White and Abdullah (1997) [61]	High level - identifies changes at the class and interface level	S13, S25, S28 -S30
T10	Vokolos and Frankl (1997) [59]	Textual Differing - Pythia	S6, S26
T11	Mansour and Fakhri (1997) [39]	Genetic algorithm	S1
T12	Mansour and Fakhri (1997) [39]	Simulated annealing	S1, S11
T13	Wong et al. (1997) [66]	Hybrid: modification, minimization and prioritization- based selection	S32
T14	Wu et al. (1999) [67]	Analysis of program structure and function-calling sequences	S33, S34
T15	Rothermel et al. (2000) [51] Harrold et al. (2001) [20] Koju et al. (2003) [34]	Edge level - identifies changes at the edge level	S8, S10, S13
T16	Orso et al. (2001) [42]	Use of metadata to represent links between changes and Test Cases	S12
T17	Sajeev et al. (2003) [55]	Use of UML (OCL) to describe information changes	S12
T18	Elbaum et al. (2003) [12]	Modified-non-core Same as T8 but ignoring core functions	S5, S22
T19	Orso et al. (2004) [43]	Partitioning and selection Two Phases	S13
T20	Pasala and Bhowmick (2005) [44]	Runtime dependencies captured and modeled into a graph (CIG)	S14
T21	Skoglund and Runeson (2005) [57]	Change based selection	S25
T22	Willmor and Embury (2005) [65]	Test selection for DB-driven applications (extension of T7) combined safety	S31
T23	Willmor and Embury (2005) [65]	Database safety	S31
T24	Mao and Lu (2005) [40]	Enhanced representation of change information	S12
T25	White et al. (2005) [62]	Extended firewall additional data-paths	S29, S30
T26	Zheng (2005) [70]	I-BACCI v.1	S35
T27	Zheng et al. (2006) [72]	I-BACCI v.2 (firewall + BACCI)	S36
T28	Zheng et al. (2006) [72]	I-BACCI v.3	S35, S36
REF1	Leung and White (1989) [36]	Retest-all	S1 - S10, S12 - S24, S26, S31 - S36
REF2		Random (25)	S7, S9
REF3		Random (50)	S7, S9
REF4		Random (75)	S7, S9
REF5		Intuitive, experience based selection	S27, S28

modules, functions or classes. There exist techniques that are not stated in their presentations to be based on the firewall technique but still make use of dependencies between software parts. T8, T14 and T18 all utilize the relations between functions and T20 use dependencies between components (DLL:s).

In addition to the three major groups, there are other techniques which share some similarities with either group, although not being directly derived from one of them.

Using the dependency principle between larger parts, such as functions or classes, lead to that all test cases using the changed part are re-executed even though the actual modified code may not be executed. Using a smaller granularity gives better precision but are usually more costly since more analysis is needed. The smallest granularity is the program statements, segments, or blocks. The relationships between these smallest parts may be represented by creating control flow graphs where the control flow from one block to another may be seen as a relationship between the two blocks. This principle is for example used in the group of techniques based on Rothermel and Harrold's technique T7, see above, but is also used in the firewall technique T5. T10 also use program blocks for its test selection. An extension of this principle where the variables are also taken into account is used in the techniques T2, T4, T6, T11-T13, in various ways.

Another group of techniques are those using specifications or metadata of the software instead of the source code or executable code. T17 use UML specifications, and T16 and T24 use metadata in XML format for their test case selection.

Uniqueness of the techniques

There is a great variance regarding the uniqueness of the techniques identified in the studied papers. Some techniques may be regarded as novel at the time of their first presentation, while others may be regarded as only variants of already existing techniques. For example in [3] a regression test selection techniques is evaluated, T8, and the technique used is based on modified entities in the subject programs. In another evaluation, reported on in [12] it is stated that the same technique is used as in [3] but adapted to use a different scope of what parts of the subjects programs that is included in the analysis, T18. In [3] the complete subject programs are included in the analysis; while in [12] core functions of the subject programs are ignored. This difference of scope probably has an effect on the test cases selected using the two different approaches. The approach in which core functions is ignored is likely to select fewer test cases compared to the approach where all parts of the programs are included. It is not obvious whether the two approaches should be regarded as two different techniques or if they should be regarded as two very similar variants of the same technique. We chose the latter option.

Some techniques evaluated in the reviewed papers are specified to be used for a specific type of software, e.g. Java, T15 and T19 [20] [43], component based soft-

ware, T17, T20, T24 and T28 [40] [44] [71] [72], or database-driven applications, T22, [65]. It is not clear whether they should be considered one technique applied to two types of software, or two distinctly different techniques. For example, a technique specified for Java, T15, is presented and evaluated in [20]. In [34] the same technique is used on MSIL (MicroSoft Intermediate Language) code, however adapted to cope with programming language constructs not present in Java. Thus, it can be argued that the results of the two studies cannot be synthesized in order to draw conclusions regarding the performance of neither the technique presented in [20], nor the adapted version, used in [34]. However, we chose to classify them as the same technique.

There are also techniques specified in a somewhat abstract manner, e.g. techniques that handle object-oriented programs in general, e.g. T14 [67]. However, when evaluating a technique, the abstract specification of a technique must be concretized to handle the specific type of subjects selected for the evaluation. The concretization may look different depending on the programming language used for the subject programs. T14 is based on dependencies between functions in object-oriented programs in general. The technique is evaluated by first tailoring the abstract specification of the technique to C++ programs and then performing the evaluation on subject programs in C++. However, it is not clear how the tailoring of the specification should be performed to evaluate the technique using other object-oriented programming languages, e.g. C# or Java. Thus, due to differences between programming languages, a tailoring made for one specific programming language may have different general performance than a tailoring made for another programming language.

3.4 Classification of Techniques (RQ2)

In response to our second research question (RQ2), we are looking for some kind of classification of the regression test selection techniques. Since the techniques are sensitive to subtle changes in their implementation or use, we could compare classes of techniques, instead of comparing individual techniques. As indicated in Figure 3, there exist many variants of techniques, gradually evolved over time. Some suggested classifications of regression test techniques exist. Rothmel and Harrold present a framework for analyzing regression test selection techniques [49], including evaluation criteria for the techniques: inclusiveness, precision, efficiency and generality. Graves et al. [16] present a classification scheme where techniques are classified as Minimization, Safe, Dataflow-Coverage-based, Ad-hoc/Random or Retest-All techniques. Orso et al. [43] separate between techniques that operate at a higher granularity e.g. method or class (called high-level) and techniques that operate at a finer granularity, e.g. statements (called low-level). In this review we searched for classifications in the papers themselves with the goal of finding common properties in order to be able to reason about groups of regression testing techniques.

One property found regards the type of input required by the techniques. The most common type of required input is source code text, e.g. T1-8, T10-12 and T18. Other types of code analyzed by techniques are intermediate code for virtual machines, e.g. T9, T13-15 and T21, or machine code, e.g. T24 and T26. Some techniques require input of a certain format, e.g. T16 (meta data) and T17 (OCL). Techniques may also be classified according to the type of code used in the analysis (Java, C++). A third type of classification that could be extracted from the papers regards the programming language paradigm. Some techniques are specified for use with procedural code, e.g. T1, T2, T7, T8, and T18, while other techniques are specified for the object-oriented paradigm, e.g. T9, T13-17, and T21-T23 some techniques are independent of programming language, e.g. T3, T19, and T26-28.

The most found property assigned to regression test selection techniques is whether they are *safe* or *unsafe*. With a safe technique the defects found with the full test suite are also found with the test cases picked by the regression test selection technique. This property may be used to classify all regression test selection techniques into either *safe* or *unsafe techniques*. Re-test all is an example of a safe technique since it selects all test cases, hence, it is guaranteed that all test cases that reveal defects are selected. Random selection of test cases is an example of an unsafe technique since there is a risk of test cases revealing defects being missed. In our study seven techniques were stated by the authors to be safe, T7, T8, T10, T15, and T21-24. However, the safety characteristic is hard to achieve in practice, as it e.g. assumes determinism in program and test execution.

A major problem, in addition to finding a classification scheme is applying the scheme to the techniques. The information regarding the different properties is usually not available in the publications. Hence, we may only give examples of techniques having the properties above based on what the authors state in their publications. The properties reported for each technique is presented in Table 8.

3.5 Analysis of the Empirical Evidence (RQ3)

Once we have defined which empirical studies exist and a list of the techniques they evaluate, we continue with the third research question on whether there are significant differences between the techniques (RQ3). We give an overview of the primary studies as such in Subsection 3.5.1. Then we focus on the metrics and evaluation criteria used in different studies (Section 3.5.2).

Types of empirical evidence

Table 11 overviews the primary studies by research method, and the size of the system used as subject. We identified 21 unique controlled experiments and 15 unique case studies. Half of the experiments are conducted on the same set of small programs [25], often referred to as the Siemens programs, which are made

Table 8: Overview of properties for each technique.

Applicability		Method		Properties			
Tech- nique	Type of Lang- uage ^a	Type of Soft- ware ^b	Input ^c	Approach ^d	Granularity ^e	Detection Ability ^f	Cost Re- duction ^g
T1	Ind		IM	CF	Stm		
T2	Proc		SC	CF	Stm		Min
T3	Proc		SC	FW	Module		
T4	Proc		SC	Slicing	Stm		Min
T5	Proc		SC	FW			
T6	Proc		SC	Slicing	Stm		
T7	Proc		SC	CF	Stm	Safe	
T8	Proc		SC	Dep	Func	Safe	
T9	OO		IM	FW	Class		
T10	Proc		SC		Stm	Safe	
T11	Proc		SC	Genetic	Stm		
T12	Proc		SC	SimAn	Stm		
T13	Proc		SC		Stm		Min
T14	OO		SC	Dep	Func		
T15	OO		IM	CF	Stm	Safe	
T16	OO	Comp	Spec	CF	Stm		
T17	OO	Comp	Spec				
T18	Proc		SC	Dep	Func		
T19	OO		IM	FW+CF	Class+Stm		
T20	Ind	Comp	BIN	Dep	Comp		
T21	OO		IM	FW	Class		
T22	OO	DB	SC	CF	Stm	Safe	
T23	OO	DB	SC	CF	Stm	Safe ^h	
T24	OO	Comp	BIN +Spec	Dep	Stm	Safe	
T25	OO		SC?	FW	Class		
T26	Ind	Comp	BIN	FW	Func		
T27	Ind	Comp	BIN+SC	FW	Func		
T28	Ind	Comp	BIN+SC	FW	Func		

^aProc= Procedural language, Ind = Independent, OO = Object oriented

^bComp = Component based, DB = Database driven

^cSC = Source code, IM = Intermediate code for virtual machines, BIN = Machine code, Spec = Input of a certain format

^dCF = Control flow, FW = Fire wall, Slicing, Dep = Dependency based, Genetic, SimAn= Simulated annealing

^eStm = statement, Func = Function, Class, Module, Component

^fSafe

^gMin = Minimization

^hsafe only in DB-state

Table 9: Primary studies of different type and size

Type of studies	Size of subjects under study	Number of studies	%
Experiment	Large	1	3
Experiment	Medium	7	19
Experiment	Small	13	36
Case study	Large	4	11
Case study	Medium	5	14
Case study	Small	4	11
Case study	Not reported	2	6
Total		36	100

available through the software infrastructure repository² presented by Do et al. [9]. The number of large scale real life evaluations is sparse. In this systematic review we found four (S25, S27, S28, S30). Both types of studies have benefits and encounter problems, and it would be of interest to study the link between them, i.e. does a technique which is shown to have great advantages in a small controlled experiment show the same advantages in a large scale case study. Unfortunately no complete link was found in this review. However, the move from small toy programs to medium sized components, which is observed among the studies, is a substantial step in the right direction towards real-world relevance and applicability.

The empirical quality of the studies varies a lot. In order to obtain a sufficiently large amount of papers, our inclusion criteria regarding quality had to be weak. Included in our analysis was any empirical evaluation of regression test selection techniques if relevant metrics were used and a sufficiently rigorous data collection and analysis could be followed in the report, see section 1.4 for more details. This was independently assessed by two researchers.

An overview of the empirically studied relations between techniques and studies are shown in Figure 4. Circles represent techniques and connective lines between the techniques represent comparative studies. CS on the lines refers to the number of case studies conducted in which the techniques are compared, and Exp denotes the number of experimental comparisons. Some techniques have not been compared to any of the other techniques in the diagram: T13, T14 and T20. These techniques are still empirically evaluated in at least one study, typically a large

² <http://sir.unl.edu>.

scale case study. If no comparison between proposed techniques is made, the techniques are compared to a reference technique instead, e.g. the retest of all test cases, and in some cases a random selection of a certain percentage of test cases is used as a reference as well. The reference techniques are not shown Figure 4 for visibility reasons.

Researchers are more apt to evaluate new techniques or variants of techniques than to replicate studies, which is clearly indicated by that we identified 28 different techniques in 27 papers. This gives rise to clusters of similar techniques compared among them selves and techniques only compared to a reference method such as re-test all.

Three clusters of techniques have been evaluated sufficiently to allow for meaningful comparison, see Figure 4; C1: T2, T7, T8 and T18, C2: T4, T5, T6 and T12, and C3: T3, T9 and T25. Each of these pair of techniques has been compared in at least two empirical studies. However, not all studies are conducted according to the same evaluation criteria, nor is the quality of the empirical evidence equally high. Therefore we classified the results with respect to empirical quality, as described in Section 1.6, and with respect to evaluation criteria, as described below.

Evaluation criteria

Do and Rothermel proposed a cost model for regression testing evaluation [10]. However, this model requires several data which is not published in the primary studies. Instead, we evaluated the results with respect to each evaluation criterion separately. We identified two main categories of metrics: *cost reduction* and *fault detection effectiveness*. Five different aspects of cost reduction and two of fault detection effectiveness have been evaluated in the primary studies. Table 12 gives an overview of the extent to which the different metrics are used in the studies. Size of test suite reduction is the most frequent, evaluated in 76% of the studies. Despite this, it may not be the most important metric. If the cost for performing the selection is too large in relation to this reduction, no savings are achieved. In 42% of the studies the total time (test selection and execution) is evaluated instead or as well. The effectiveness measures are either related 1) to test cases, i.e. the percentage of fault-revealing test cases selected out of all fault-revealing test cases, or 2) to faults, i.e. the percentage of faults out of all known ones, detected by the selected test cases.

Several of the studies concerning reduction of number of test cases are only compared to retest all (S8, S10, S14-S21, S26, S32-S34) [20], [34], [44], [50], [52], [60], [66], [67] with the only conclusion that a reduction of test cases can be achieved, but nothing on the size of the effect in practice. This is a problem identified in experimental studies in general [28]. Many of the studies evaluating time reduction are conducted on small programs, and the size of the differences is measured in milliseconds, although there is a positive trend, over time, towards using medium-sized programs. Only 30% of the studies consider both fault detection

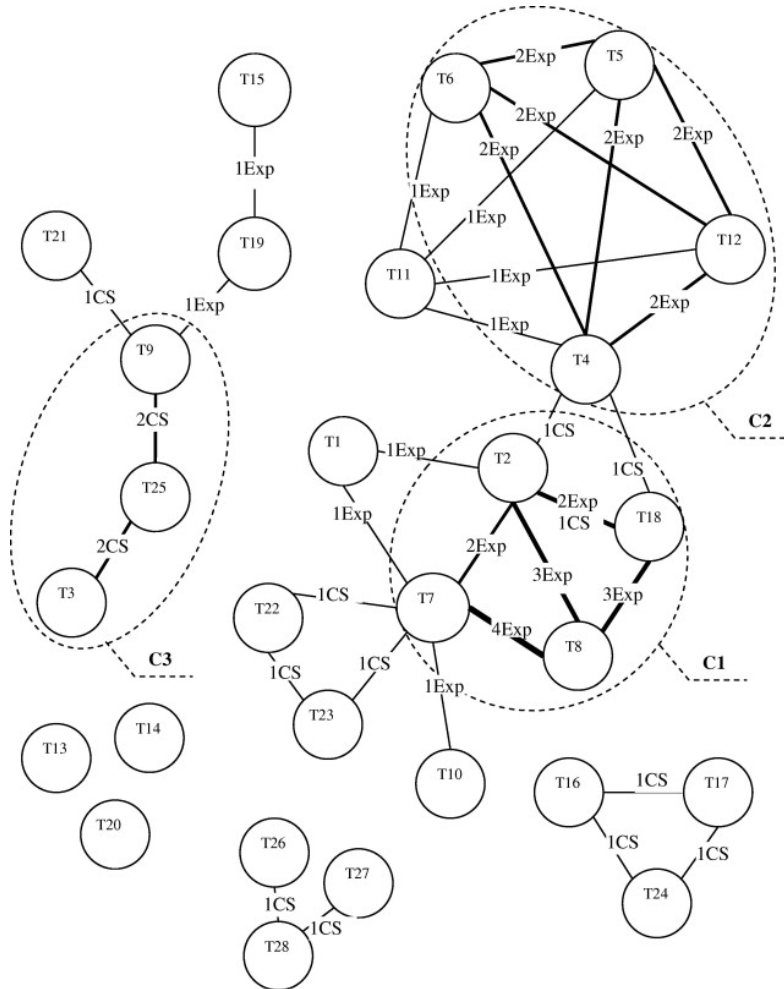


Figure 4: Techniques related to each other through empirical comparisons.

and cost reduction. Rothmel proposed a framework for evaluation of regression test selection techniques [49] which have been used in some evaluations. This framework defines four key metrics, inclusiveness, precision, efficiency, and generality. Inclusiveness and precision corresponds to test case-related fault detection effectiveness and precision, respectively, in Table 10. Efficiency is related to space and time requirements and varies with test suite reduction as well as with test execution time and test selection time. Generality is more of a theoretical reasoning, which is not mirrored in the primary studies.

Table 10: Use of evaluation metrics in the studies

Evaluated Metrics		Number	%	Rothermel framework [49]
Cost Reduction	Test suite reduction	29	76	Efficiency
	Test execution time	7	18	Efficiency
	Test selection time	5	13	Efficiency
	Total time	16	42	Efficiency
	Precision (omission of non-fault revealing tests)	1	3	Precision
Fault Detection Effectiveness	Test case-related detection effectiveness	5	13	Inclusiveness
	Fault-related detection effectiveness	8	21	

3.6 Comparison of techniques (RQ4)

In response to our fourth research question (RQ4) we are analyzing the empirically evaluated relations between the techniques by visualizing the results of the studies. Due to the diversity in evaluation criteria and in empirical quality this visualization cannot give a complete picture. However, it may provide answers to specific questions: e.g. Is there any technique applicable in my context proven to reduce testing costs more than the one I use today?

Our taxonomy for analyzing the evidence follows the definitions in Table 2. Grey arrows indicate *light weight* empirical result and black arrows indicate *medium weight* result. A connection without arrows in the figures means that the studies have similar effect, while where there is a difference, the arrow points to the technique that is better with respect to the chosen criterion. A connection with thicker line represents more studies. In section 3.6.1, we report our findings regarding test suite reduction and in section 3.6.2 regarding fault detection. Note that the numbers on the arrows indicate number of collected metrics, which may be more than one per study.

Cost reduction

Figure 5 reports the empirically evaluated relations between the techniques regarding the cost reduction, including evaluations of execution time as well as of test suite reduction and precision.

The strongest evidence can be found in cluster C1, where T2 provides most reduction of execution costs. T7, T8 and T18 reduce the test suites less than T2, and T8 among those reduces execution cost less than T18. All techniques how-

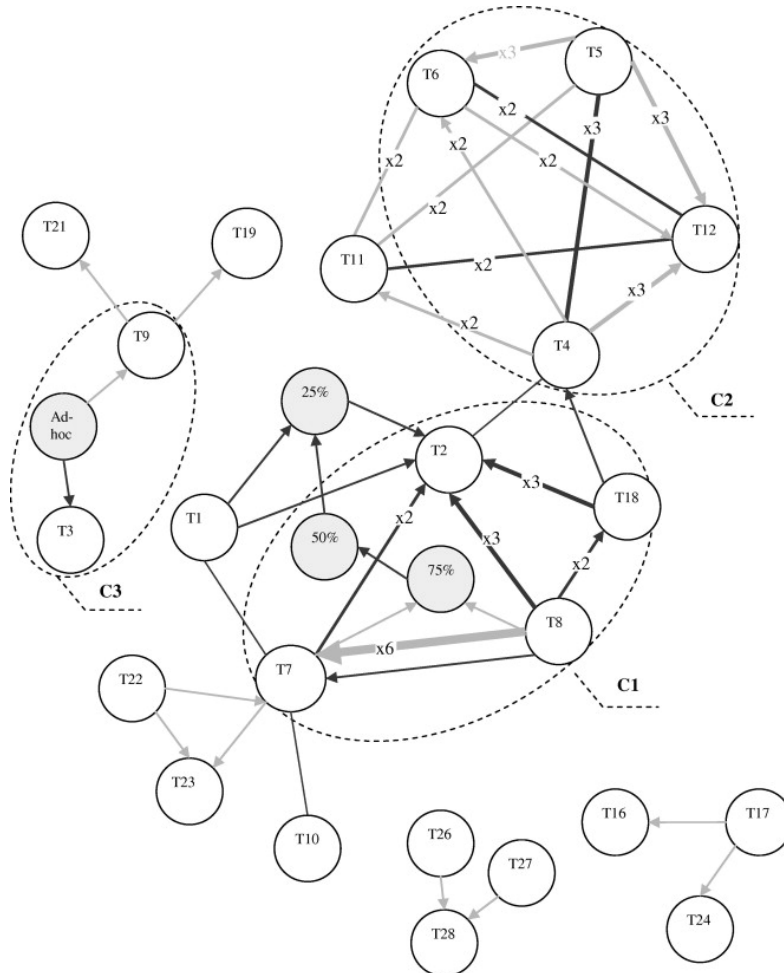


Figure 5: Empirical results for Cost Reduction, including Test Execution Time, Test Suite Reduction and Precision.

ever, reduce test execution cost compared to REF1 (re-test all), which is a natural criterion for a regression test selection technique.

In cluster C2, there is strong evidence that T6 and T12 have similar cost for test execution. On the other hand, there is a study with weaker empirical evidence, indicating that T12 reduces execution cost more than T6.

The rest of the studies show rather weak empirical evidence, showing that the evaluated techniques reduce test execution cost better than re-test all.

One component of the cost for regression test selection is the analysis time

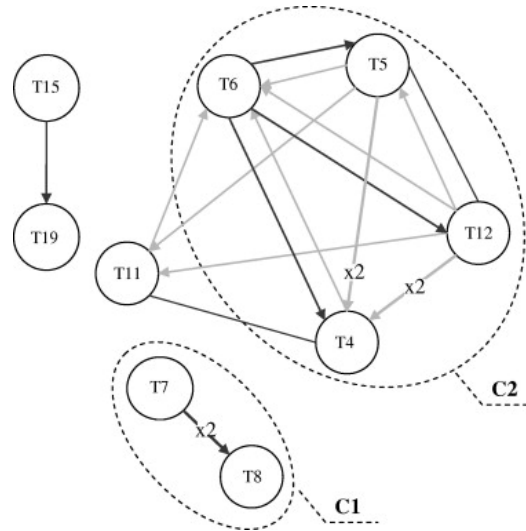


Figure 6: Empirical results for Test Selection Time.

needed to select which test cases to re-execute. The selection time is reported separately for a small subset of the studies, as shown in Figure 6.

The left group primarily tells that T19 has less selection time than T15, and in C1, T8 has less analysis time than T7.

The results from cluster C2 shows mixed messages. T4 has in most cases the shortest selection time, although it in one study is more time consuming than T6. The selection time is hence dependent on the subject programs, test cases and types of changes done.

In Figure 7, the total time for analysis and execution together is shown for those studies where it is reported. It is worth noting that some regression test selection techniques actually can be more time consuming than re-test all (T7, T8, T10). Again, this is case dependent, but it is interesting to observe that this situation actually arises under certain conditions.

Other relations are a natural consequence of the expansion of certain techniques. T9 (Object oriented firewall) is less time consuming than T25 (extended OO firewall with data paths). Here an additional analysis is conducted in the regression test selection.

Fault detection effectiveness

In addition to saving costs, regression test selection techniques should detect as many as possible of the faults found by the original test suite. Evaluations of

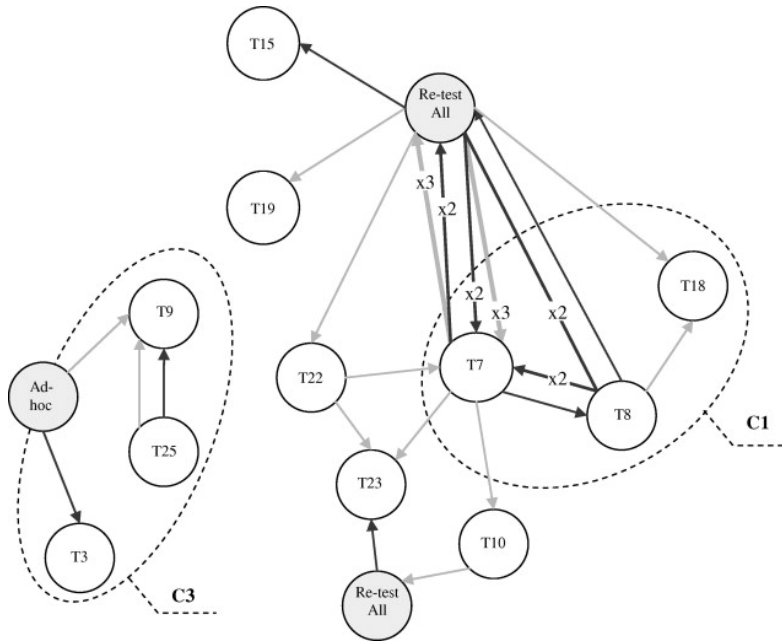


Figure 7: Empirical results for Total Time.

test case-related as well as fault-related detection effectiveness are presented in Figure 8.

Some techniques are proven to be *safe*, i.e. guarantees that the fault detection effectiveness is 100% compared to the original test suite (see Section 2.4). This property is stated to hold for seven techniques: T7, T8, T10, T15, T22, T23 and T24.

T7 and T8 within C2 are also those that can be found superior or equal from Fig 8, which is in line with the safe property. T4 in C2 tends also to be better or equal to all its reference techniques. However, for the rest, the picture is not clear.

4 Discussion

4.1 The reviewed studies

The overall goal with the study was to identify regression test selection techniques and systematically assess the empirical evidence collected about those techniques. As the selection of a specific technique is dependent on many factors, the outcomes of empirical studies also depend on those factors. However only few factors are specifically addressed in the empirical studies and hence it is not possible to draw

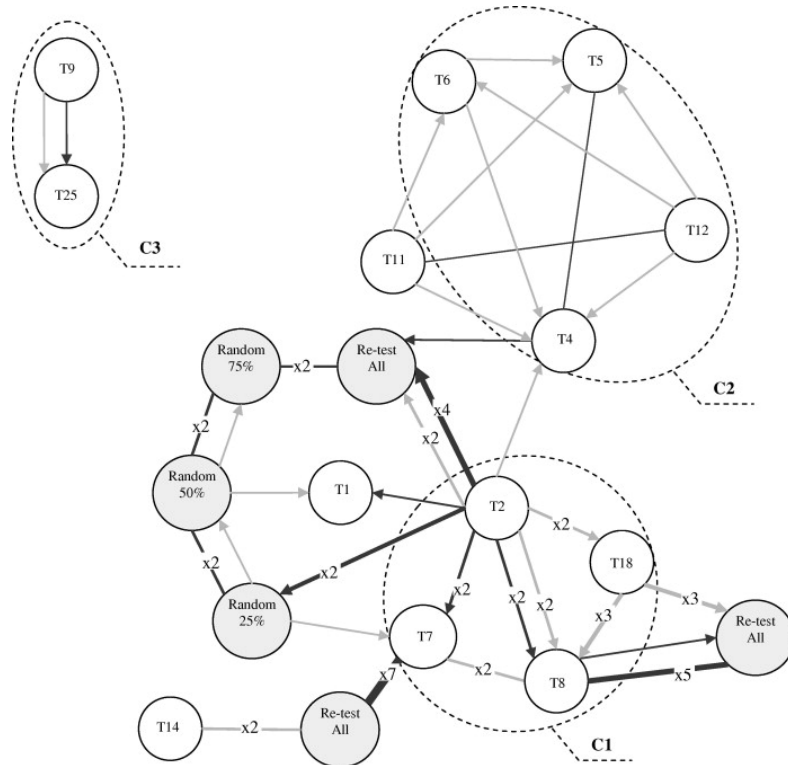


Figure 8: Empirical results for Fault Detection Effectiveness.

very precise conclusions. Nor is it possible to draw general conclusions. Instead we have conducted mostly qualitative assessments of the empirical studies. From those we try to aggregate recommendations of which regression test selection techniques to use.

A comparison of the techniques in cluster C1 indicates that the minimization technique, T2, is the most efficient in reducing time and/or number of test cases to run. However this is an unsafe technique (see Section 2.4) and all but one of six studies report on significant losses in fault detection. When it comes to safe techniques, T7 is shown to be the most efficient in reducing test cases. However analysis time for T7 is shown to be too long (it exceeds the time for rerunning all test cases) in early experiments, while in later experiments, it is shown to be good. Hence, there is a trade-off between cost reduction and defect detection ability. This is the case in all test selection, and none of the evaluated technique seems to have done any major breakthrough in solving this trade-off.

It is interesting to notice that the technique T7 is not changed between the

studies that show different results on selection time, but the subject programs on which the experiments are conducted are changed. This is one factor that heavily impacts on the performance of some techniques. This emphasizes the importance of the regression testing context in empirical studies, and may also imply that specific studies have to be conducted when selecting a technique for a specific environment.

As mentioned before, many techniques are incremental improvements of existing techniques, which are demonstrated to perform better. For example, T25 is an extension of T9, with better fault detection at the cost of total time. This is a pattern shown in many of the studies: improvements may be reached, but always at a price for something else.

4.2 Implications for future studies

The standards for conducting empirical studies, and which measures to evaluate, differ greatly across the studies. Rothermel and Harrold proposed a framework to constitute the basis for comparison [49], but it is not used to any significant level in later research. Hence, it is not possible to conduct very strict aggregation of research results, e.g. through meta analysis. It is however not necessarily the ultimate goal to compare specific techniques. More general concepts would be more relevant to analyze, rather than detailed implementation issues.

Examples of such concepts to evaluate are indicated in the headings of Table 10. *Applicability*: are different techniques better suited for different languages or programming concepts, or for certain types of software? *Method*: are some selection approaches better suited to find faults, independently of details in their implementation? Which level of granularity for the analysis is effective - statement, class, component, or even specification level? Other concepts are related to process, product and resources factors [54]. *Process*: How frequent should the regression testing cycles be? At which testing level is the regression testing most efficient: unit, function, system? *Product*: Is regression testing different for different types and sizes of products? *Resources*: Is the regression testing different with different skills and knowledge among the testers?

In the reviewed studies, some of these aspects are addressed: e.g. the size aspect, scaling up from small programs to medium-sized [52], the level of granularity of tests [3], as well as testing frequency [29] and the effect of changes [12]. However, this has to be conducted more systematically by the research community.

Since the outcomes of the studies depend on many different factors, replication of studies with an attempt to keep as many factors stable as possible is a means to achieve a better empirical foundation for evaluation of concepts and techniques. The use of benchmarking software and test suites is one way of keeping factors stable between studies [9] However, in general, the strive for novelty in each research contribution tends to lead to a lack of replications and thus a lack of deeper understanding of earlier proposed techniques.

A major issue in this review is to find the relevant information to compare techniques. Hence, for the future, a more standardized documentation scheme would be helpful, as proposed by e.g. Jedlitschka and Pfahl [26] for experiments and Runeson and Höst [53] for case studies. To allow enough detail despite page restrictions, complementary technical reports could be published on the empirical studies.

5 Conclusions and future work

In this paper we present results from a systematic review of empirical evaluations of regression test selection techniques. Related to our research questions we have identified that:

- RQ1** There are 28 empirically evaluated techniques on regression test selection published,
- RQ2** these techniques might be classified according to: applicability on type of software and type of language; details regarding the method such as which input is required, which approach is taken and on which level of granularity is changes considered; and properties such as classification in safe/unsafe or minimizing/not minimizing.
- RQ3** The empirical evidence for differences between the techniques is not very strong, and sometimes contradictory, and
- RQ4** hence there is no basis for selecting one superior technique. Instead techniques have to be tailored to specific situations, e.g. initially based on the classification of techniques.

We have identified some basic problems in the regression testing field which hinders a systematic review of the studies. Firstly, there is a great variance in the uniqueness of the techniques identified. Some techniques may be presented as novel at the time of their publications and others may be regarded as variants of already existing techniques. Combined with a tendency to consider replications as second class research, the case for cooperative learning on regression testing techniques is not good. In addition to this, some techniques are presented in a rather general manner, e.g. claimed to handle object-oriented programs, which gives much space for different interpretations on how they may be implemented due to e.g. different programming language constructs existing in different programming languages. This may lead to different (but similar) implementations of a specific technique in different studies depending on e.g. the programming languages used in the studies.

As mentioned in Section 1, to be able to select a strategy for regression testing, relevant empirical comparisons between different methods are required. Where

such empirical comparisons exist, the quality of the evaluations must be considered. One goal of this study was to determine whether the literature on regression test selection techniques provides such uniform and rigorous base of empirical evidence on the topic that makes it possible to use it as a base for selecting a regression test selection method for a given software system.

Our study shows that most of the presented techniques are not evaluated sufficiently for a practitioner to make decisions based on research alone. In many studies, only one aspect of the problem is evaluated and the context is too specific to be easily applied directly by software developers. Few studies are replicated, and thus the possibility to draw conclusions based on variations in test context is limited. Of course even a limited evidence base could be used as guidance. In order for a practitioner to make use of these results, the study context must be considered and compared to the actual environment into which a technique is supposed to be applied.

Future work for the research community is 1) focus more on general regression testing concepts rather than on variants of specific techniques; 2) encourage systematic replications of studies in different context, preferably with a focus on gradually scaling up to more complex environments; 3) define how empirical evaluations of regression test selection techniques should be reported, which variation factors in the study context are important.

Acknowledgements

The authors acknowledge Dr. Carina Andersson for her contribution to the first two stages of the study. The authors are thankful to librarian Maria Johnsson for excellent support in the search procedures. We appreciate review comments from Prof. Sebastian Elbaum and the anonymous reviewers, which substantially have improved the paper. The work is partly funded by the Swedish Governmental Agency for Innovation Systems under grant 2005-02483 for the UPPREPA project, and partly funded by the Swedish Research Council under grant 622-2004-552 for a senior researcher position in software engineering.

References

- [1] Hiralal Agrawal, Joseph R. Horgan, Edward W. Krauser, and Saul A. London. Incremental regression testing. In *Proceedings of the Conference on Software Maintenance*, pages 348–357, 1993.
- [2] Ghinwa Baradhi and Nashat Mansour. A comparative study of five regression testing algorithms. In *Proceedings of the Australian Software Engineering Conference (ASWEC '97)*, pages 174–182, 1997.
- [3] John Bible, Gregg Rothermel, and David S. Rosenblum. A comparative study of coarse- and fine-grained safe regression test-selection techniques. *ACM Transactions on Software Engineering and Methodology*, 10(2):149–183, 2001.
- [4] David Binkley. The application of program slicing to regression testing. *Information and Software Technology*, 40(11-12):583–594, 1998.
- [5] Pearl Brereton, Barbara A. Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software*, 80(4):571–583, April 2007.
- [6] Yanping Chen, Robert L. Probert, and D. Paul Sims. Specification-based regression test selection with risk analysis. In *Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research (CASCON '02)*, pages 1–. IBM Press, 2002.
- [7] Yih-Farn Chen, David S. Rosenblum, and Kiem-Phong Vo. TESTTUBE: a system for selective regression testing. In *Proceedings of the 16th International Conference on Software Engineering (ICSE-16)*, pages 211 –220, May 1994.
- [8] Oscar Dieste, Anna Grimán, and Natalia Juristo. Developing search strategies for detecting relevant experiments. *Empirical Software Engineering*, 14(5):513–539, October 2009.
- [9] Hyunsook Do, Sebastian Elbaum, and Gregg Rothermel. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Software Engineering*, 10(4):405–435, 2005.
- [10] Hyunsook Do and Gregg Rothermel. An empirical study of regression testing techniques incorporating context and lifetime factors and improved cost-benefit models. In *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering (FSE-14)*, pages 141–151, 2006.

-
- [11] Tore Dybå, Torgeir Dingsøy, and Geir K. Hanssen. Applying systematic reviews to diverse study types: An experience report. In *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement (ESEM '07)*, pages 225–234, 2007.
- [12] Sebastian Elbaum, Praveen Kallakuri, Alexey Malishevsky, Gregg Rothermel, and Satya Kanduri. Understanding the effects of changes on the cost-effectiveness of regression testing techniques. *Software Testing, Verification and Reliability*, 13(2):65–83, April 2003.
- [13] Emelie Engström, Mats Skoglund, and Per Runeson. Empirical evaluations of regression test selection techniques: a systematic review. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, ESEM '08, pages 22–31, New York, NY, USA, 2008. ACM.
- [14] Kurt F. Fischer, F Raji, and A Chruscicki. A methodology for retesting modified software. In *Proceedings of the National Telecommunications Conference*, pages 1–6, 1981.
- [15] Phyllis G. Frankl, Gregg Rothermel, Kent Sayre, and Filippos I. Vokolos. An empirical comparison of two safe regression test selection techniques. In *Proceedings of the International Symposium on Empirical Software Engineering (ISESE 2003)*, pages 195 – 204, October 2003.
- [16] Todd L. Graves, Mary Jean Harrold, Jung-Min Kim, Adam Porter, and Gregg Rothermel. An empirical study of regression test selection techniques. *ACM Transactions on Software Engineering and Methodology*, 10(2):184–208, April 2001.
- [17] Rajiv Gupta, Mary Jean Harrold, and Mary Lou Soffa. An approach to regression testing using slicing. In *In Proceedings of the Conference on Software Maintenance*, pages 299–308, 1992.
- [18] Rajiv Gupta, Mary Jean Harrold, and Mary Lou Soffa. Program slicing-based regression testing techniques. *Journal of Software Testing, Verification, and Reliability*, 6(2):83–111, 1996.
- [19] Florian Haftmann, Donald Kossmann, and Eric Lo. A framework for efficient regression tests on database applications. *The VLDB Journal*, 16(1):145–164, September 2006.
- [20] Mary Jean Harrold, Alessandro Orso, James A. Jones, Maikel Pennings, Tongyu Li, Ashish Gujarathi, Saurabh Sinha, Donglin Liang, and S. Alexander Spoon. Regression test selection for java software. In *Proceedings of the 16th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA '01)*, pages 312–326, 2001.

- [21] Mary Jean Harrold and Mary Lou Soffa. An incremental approach to unit testing during maintenance. In *Proceedings of the Conference on Software Maintenance*, pages 362–367, October 1988.
- [22] Jean Hartmann and David J. Robson. Approaches to regression testing. In *Proceedings of the Conference on Software Maintenance*, pages 368–372, October 1988.
- [23] Jean Hartmann and David J. Robson. Techniques for selective revalidation. *IEEE Software*, 7(1):31–36, January 1990.
- [24] Pei Hsia, Xiaolin Li, David Chenho Kung, Chih-Tung Hsu, Liang Li, Yasufumi Toyoshima, and Cris Chen. A technique for the selective revalidation of OO software. *Journal of Software Maintenance*, 9(4):217–233, July 1997.
- [25] Monica Hutchins, Herb Foster, Tarak Goradia, and Thomas Ostrand. Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria. In *Proceedings of the 16th international conference on Software engineering (ICSE '94)*, pages 191–200, 1994.
- [26] Andreas Jedlitschka and Dietmar Pfahl. Reporting guidelines for controlled experiments in software engineering. In *2005 International Symposium on Empirical Software Engineering (ISESE 2005)*, pages 95–104, November 2005.
- [27] Natalia Juristo, Ana Moreno, Sira Vegas, and Martin Solari. In search of what we experimentally know about unit testing. *IEEE Software*, 23(6):72–80, November 2006.
- [28] Vigdis By Kampenes, Tore Dybå, Jo E. Hannay, and Dag I.K. Sjøberg. A systematic review of effect size in software engineering experiments. *Information and Software Technology*, 49(11-12):1073–1086, November 2007.
- [29] Jung-Min Kim, Adam Porter, and Gregg Rothermel. An empirical study of regression test application frequency. *Software Testing, Verification and Reliability*, 15(4):257–279, December 2005.
- [30] B.A. Kitchenham, E. Mendes, and G.H. Travassos. Cross versus within-company cost estimation studies: A systematic review. *Software Engineering, IEEE Transactions on*, 33(5):316–329, May 2007.
- [31] Barbara Kitchenham. Guidelines for performing systematic literature reviews in software engineering. Technical report, and Department of Computer Science, University of Durham, Version 2.3, 2007.

-
- [32] Barbara Kitchenham, O. Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. Systematic literature reviews in software engineering - a systematic literature review. *Information and Software Technology*, 51(1):7–15, January 2009.
- [33] René R. Klösch, Paul W. Glaser, and Robert J. Truschneegg. A testing approach for large system portfolios in industrial environments. *Journal of Systems and Software*, 62(1):11–20, May 2002.
- [34] Toshihiko Koju, Shingo Takada, and Norihisa Doi. Regression test selection based on intermediate code for virtual machines. In *Proceedings of the International Conference on Software Maintenance (ICSM '03)*, pages 420–429, 2003.
- [35] J. Richard Landis and Gary G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159–174, March 1977.
- [36] Hareton K. N. Leung and Lee White. Insights into testing and regression testing global variables. *Journal of Software Maintenance*, 2(4):209–222, December 1990.
- [37] Hareton K. N. Leung and Lee White. A study of integration testing and software regression at the integration level. In *Proceedings of the IEEE Conference on Software Maintenance*, pages 290–301, November 1990.
- [38] Nashat Mansour, Rami Bahsoon, and Ghinwa Baradhi. Empirical comparison of regression test selection algorithms. *Journal of Systems and Software*, 57(1):79–90, April 2001.
- [39] Nashat Mansour and Khaled El-Fakih. Natural optimization algorithms for optimal regression testing. In *Proceedings of the 21th Annual International Computer Software and Applications Conference (COMPSAC '97)*, pages 511–514, August 1997.
- [40] Chengying Mao and Yansheng Lu. Regression testing for component-based software systems by enhancing change information. In *Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC '05)*, pages 611–618, 2005.
- [41] Atif M. Memon. Using tasks to automate regression testing of GUIs. In *Proceedings of the International Conference Applied Informatics (IASTED)*, pages 477–482, 2004. cited By (since 1996) 2.
- [42] Alessandro Orso, Mary Jean Harrold, David Rosenblum, Gregg Rothermel, Hyunsook Do, and Mary Lou Soffa. Using component metacontent to support the regression testing of component-based software. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01)*, pages 716–725, 2001. cited By (since 1996) 33.

- [43] Alessandro Orso, Nanjuan Shi, and Mary Jean Harrold. Scaling regression testing to large software systems. In *Proceedings of the ACM SIGSOFT 12th International symposium on Foundations of software engineering (FSE-12)*, volume 29, pages 241–251, October 2004.
- [44] Anjaneyulu Pasala and Animesh Bhowmick. An approach for test suite selection to validate applications on deployment of COTS upgrades. In *Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC '05)*, pages 401–407, 2005.
- [45] Xiaoxia Ren, Fenil Shah, Frank Tip, Barbara G. Ryder, and Ophelia Chesley. Chianti: a tool for change impact analysis of java programs. *Proceedings of the Object-Oriented Programming Systems, Languages, and Applications (OOPSLA 2004)*, 39(10):432–448, October 2004.
- [46] 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 *Proceedings of the 24th International Conference on Software Engineering (ICSE '02)*, pages 130–140, 2002.
- [47] Gregg Rothermel, Sebastian Elbaum, Alexey G. Malishevsky, Praveen Kallakuri, and Xuemei Qiu. On test suite composition and cost-effective regression testing. *ACM Trans. Softw. Eng. Methodol.*, 13(3):277–331, July 2004.
- [48] Gregg Rothermel and Mary Jean Harrold. A safe, efficient algorithm for regression test selection. In *Proceedings of the Conference on Software Maintenance (ICSM '93)*, pages 358–367, 1993.
- [49] Gregg Rothermel and Mary Jean Harrold. Analyzing regression test selection techniques. *IEEE Transactions on Software Engineering*, 22(8):529–551, August 1996.
- [50] Gregg Rothermel and Mary Jean Harrold. A safe, efficient regression test selection technique. *ACM Transactions on Software Engineering and Methodology*, 6(2):173–210, April 1997.
- [51] Gregg Rothermel, Mary Jean Harrold, and Jainay Dedhia. Regression test selection for c++ software. *Software Testing, Verification and Reliability*, 10(2):77–109, June 2000.
- [52] Gregg Rothermel, Mary Jean Harrold, Jeffery Ostrin, and Christie Hong. An empirical study of the effects of minimization on the fault detection capabilities of test suites. In *Proceedings of the IEEE International Conference on Software Maintenance*, page 34, 1998.

-
- [53] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, April 2009.
- [54] Per Runeson, Mats Skoglund, and Emelie Engström. Test benchmarks—what is the question? In *Proceedings of the IEEE International Conference on Software Testing Verification and Validation Workshop (ICSTW '08)*, pages 368–371, April 2008.
- [55] A. S. M. Sajeev and Bugi Wibowo. Regression test selection based on version changes of components. In *Proceedings of the IEEE Tenth Asia-Pacific Software Engineering Conference (APSEC '03)*, pages 78–85, 2003.
- [56] William R. Shadish, Thomas D. Cook, and Donald T. Campbell. *Experimental and Quasi-Experimental Designs for Generalized Causal Inference*. Houghton Mifflin Company, Boston New York, 2002.
- [57] Mats Skoglund and Per Runeson. A case study of the class firewall regression test selection technique on a large scale distributed software system. In *International Symposium on Empirical Software Engineering*, pages 72–81, 2005.
- [58] Mark Staples and Mahmood Niazi. Experiences using systematic review guidelines. *Journal of Systems and Software*, 80(9):1425–1437, September 2007.
- [59] Filippos I. Vokolos and Phyllis G. Frankl. Pythia: A regression test selection tool based on textual differencing. In *Proceedings of the Third International Conference on Reliability, Quality and Safety of Software-Intensive Systems (ENCRESS'97)*, 1997.
- [60] Filippos I. Vokolos and Phyllis G. Frankl. Empirical evaluation of the textual differencing regression testing technique. In *Software Maintenance, 1998. Proceedings. International Conference on*, pages 44–53. IEEE Comput. Soc, November 1998.
- [61] Lee White and Khalil Abdullah. A firewall approach for the regression testing of object-oriented software. In *Proceedings of 10th annual software quality week*, 1997.
- [62] Lee White, Khaled Jaber, and Brian Robinson. Utilization of extended firewall for object-oriented regression testing. In *Proceedings of the IEEE 21st International Conference on Software Maintenance (ICSM '05)*, pages 695–698, 2005.
- [63] Lee White and Brian Robinson. Industrial real-time regression testing and analysis using firewalls. In *Proceedings of the 20th IEEE International Conference on Software Maintenance (ICSM '04)*, pages 18–27, 2004.

- [64] Lee J. White and Hareton K.N. Leung. A firewall concept for both control-flow and data-flow in regression integration testing. In *Software Maintenance, 1992. Proceedings., Conference on*, pages 262–271. IEEE Comput. Soc. Press, November 1992.
- [65] David Willmor and Suzanne M. Embury. A safe regression test selection technique for database driven applications. In *Proceedings of the IEEE International Conference on Software Maintenance*, pages 421–430, 2005.
- [66] W. Eric Wong, Joseph R. Horgan, Saul London, and Hira Agrawal Bellcore. A study of effective regression testing in practice. In *Proceedings of the IEEE Eighth International Symposium on Software Reliability Engineering (ISSRE '97)*, pages 264–274, 1997.
- [67] Ye Wu, Mei-Hwa Chen, and H.M. Kao. Regression testing on object-oriented programs. In *Proceedings of the 10th International Symposium on Software Reliability Engineering*, pages 270–279, 1999.
- [68] Robert K. Yin. *Case Study Research: Design and Methods*. SAGE, 2003.
- [69] Jiang Zheng. In regression testing selection when source code is not available. In *Proceedings of the IEEE/ACM 20th international Conference on Automated software engineering (ASE '05)*, pages 752–755, 2005.
- [70] Jiang Zheng, Brian Robinson, Laurie Williams, and Karen Smiley. An initial study of a lightweight process for change identification and regression test selection when source code is not available. In *Proceedings of the IEEE 16th International Symposium on Software Reliability Engineering (ISSRE 2005)*, pages 225–234, November 2005.
- [71] Jiang Zheng, Brian Robinson, Laurie Williams, and Karen Smiley. Applying regression test selection for COTS-based applications. In *Proceedings of the 28th international conference on Software engineering (ICSE '06)*, pages 512–522, 2006.
- [72] Jiang Zheng, Brian Robinson, Laurie Williams, and Karen Smiley. A lightweight process for change identification and regression test selection in using COTS components. In *Proceedings of the Fifth International Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems (ICCBSS '06)*, pages 137–143, 2006.

A QUALITATIVE SURVEY OF REGRESSION TESTING PRACTICES

Abstract

Aim: Regression testing practices in industry have to be better understood, both for the industry itself and for the research community. *Method:* We conducted a qualitative industry survey by i) running a focus group meeting with 15 industry participants and ii) validating the outcome in an on line questionnaire with 32 respondents. *Results:* Regression testing needs and practices vary greatly between and within organizations and at different stages of a project. The importance and challenges of automation is clear from the survey. *Conclusions:* Most of the findings are general testing issues and are not specific to regression testing. Challenges and good practices relate to test automation and testability issues.

Emelie Engström and Per Runeson

Proceedings of *11th International Conference on Product Focused Software Development and Process Improvement (PROFES'10)*, June 2010

1 Introduction

Regression testing is retesting of previously working software after a change to ensure that unchanged software is still functioning as before the change. According to IEEE, regression testing is *Selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or components still complies with its specified requirements* [8]. The need for effective strategies for regression testing increases with the increasing use of iterative development strategies and systematic reuse in software projects. Studies indi-

cate that 80% of testing cost is regression testing and more than 50% of software maintenance cost is related to testing [3].

There is a gap between research and practices of regression testing. Research on regression testing mainly focuses on selection and prioritization of test cases. Several techniques for regression test selection are proposed and evaluated. Engström *et al.* reviewed the literature in the field recently [4] and highlights the importance of the test context to the outcome of regression testing techniques. Only few empirical evaluations of regression test selection techniques are carried out in a real industrial context [5], [16], [17].

However industry practice on regression testing is mostly based on experience alone, and not on systematic approaches. There is a need for researchers to better understand the needs and practices in industry. Rooksby *et al.* [11] argue for the need for investigation and characterization of real world work. They conclude that improvements of current testing practices are meaningful in its specific local context and “cannot be brought about purely through technically driven innovation”. In their paper they highlight, based on experiences from testing in four real projects, that improvements in industry are not always sophisticated and accurate as is often pursued in research.

In order to retrieve a better understanding of real world needs and practices, a qualitative survey [6, p. 61-78] of industry practice of regression testing is conducted, by means of focus group discussions in a software process improvement network (SPIN) and a questionnaire to validate the results. Issues discussed in the focus group were definitions and practices of regression testing in industry as well as challenges and improvement suggestions. A total of 46 software engineers from 38 different organizations participated in the focus group and questionnaire survey. Results are qualitative and of great value in that they highlight relevant and possible directions for future research.

To the extent of our knowledge no industrial surveys on regression testing practices have been reported on. However experience reports on regression testing in industrial software development projects can be found [9]. Onoma *et al.* conclude that regression testing is used extensively and that several companies develop in-house regression testing tools to automate the process. Re-test all is a common approach and the selection of test cases is not a critical issue.

When it comes to testing practices in general a couple of industrial surveys have been undertaken [2], [7], [12], [13], concluding that test automation is a key improvement issue [13] and that test case selection for continuous regression testing is a hard task. No systematic approach for test case selection was used by the companies but instead they relied on the developers expertise and judgment [12].

This paper is organized as follows: Section 2 describes how the survey is conducted and discusses validity issues. In section 3 results are presented and analyzed. Finally conclusions are provided in section 4.

2 Method description

The study's overall goal is to characterize current regression testing practices in industry for the sake of research. It also aims at identifying good practices for spreading across different companies as well as areas in need for improvement within the companies and possibly identification of future research topics. Hence, a qualitative survey is found appropriate [6, p. 61-78]. The research questions for the survey are:

RQ1 What is meant by *regression testing* in industry?

RQ2 Which *problems* or *challenges* related to regression testing exist?

RQ3 Which *good practices* on regression testing exist?

The survey is conducted using two different research methods, one focus group discussion [10, p. 284-289] in a SPIN group, and one questionnaire in a testing interest network. The focus group was used to identify concepts and issues related to regression testing, while the questionnaire was used to validate the findings in a different setting. A similar approach was used for a unit testing survey in 2006 [12].

2.1 Focus group

The focus group meeting was arranged at one of the monthly meetings of SPIN-syd, a software process improvement network in Southern Sweden [14]. The members of the network were invited to a 2.5 hour session on regression testing in May 2009. 15 industry participants accepted the invitation, which is about the normal size for a SPIN-syd monthly meeting, and the same as for our previous unit testing survey [12]. The focus group meeting was moderated by two academics and one industry participant, and observed by a third academic. An overview of the focus group participants is shown in Table 1.

The industry participants represented automation, medical devices, information systems (IS), and telecom domains. Consultants also participated which were working with testing for their clients. The product companies all produce embedded software and were both of medium and large size, while consultancy firms of all sizes were represented.

The session was organized around five questions:

- What is regression testing?
- When do the participants regression test?
- How do the participants regression test?
- What are the participants' problems regarding regression testing?

Table 1: Participants in focus group meeting. Number of developers in the surveyed company: extra small is 1, small is 2 – 19, medium is 20 – 99, and large 100 – 999

Company	Domain	Size	Role
A	Automation	Medium	Participant
A	Automation	Medium	Participant
A	Automation	Medium	Participant
G	Medical devices	Medium	Participant
G	Medical devices	Medium	Participant
I	Information systems	Large	Moderator
I	Information systems	Large	Participant
S	Telecom	Large	Participant
S	Telecom	Large	Participant
E	Telecom	Large	Participant
X	Consultant	Extra small	Participant
C	Consultant	Extra small	Participant
Q	Consultant	Medium	Participant
K	Consultant	Medium	Participant
O	Consultant	Large	Participant
L	Academics	N/A	Researcher
L	Academics	N/A	Researcher
L	Academics	N/A	Observer

- What are the participants' strengths regarding regression testing?

For each of the questions, the moderator asked the participants to write their answers on post-it charts. Then each participant presented his or her view of the question and the responses were documented on white boards.

After the session, key findings were identified using qualitative analysis methods. Statements were grouped into themes, primarily structured by the five questions, and secondary according to keywords in the statements. Further, the results were restructured and turned into questions for use in the questionnaire.

2.2 Questionnaire

The resulting questionnaire consists of 45 questions on what regression testing is, with five-level Likert-scale response alternatives: *Strongly disagree*, *Disagree*, *Neutral*, *Agree*, *Strongly Agree* and an additional *Not Applicable* option (see Fig 1). One question on automation vs manual used five scale alternatives from *Automated* to *Manual* (see Fig 2). Further, 29 questions on satisfaction with regression testing practices in the respondents' organizations had the response alternatives *Very Satisfied*, *Satisfied*, *Neutral*, *Dissatisfied*, *Very Dissatisfied* and *Not Applicable* (see Fig 3). The questionnaire was defined in the SurveyGizmo questionnaire tool for on line data collection [1].

Respondents were invited through the SAST network (Swedish Association for Software Testing) through their quarterly newsletter, which is distributed to some 2.000 testers in Sweden, representing a wide range of company sizes and application domains. Respondents were promised an individual benchmarking report if more than three participants from one company responded, and a chance for everybody to win a half-day seminar on testing given by the second author. Thirty-two respondents answered the complete questionnaire, which are presented in Table 2.

The respondents cover the range of company sizes and domains. Out of the 32 respondents, 9 were developing embedded systems in particular within the telecom domain, 12 developed information systems in particular within the domains of business intelligence and finance, and 11 were consultants. Out of 21 product companies, 3 represent small development organizations, 9 represent medium sized organizations and 8 represent large organizations. The size of the consultancy organizations are not specifically relevant, but is reported to indicate the variation.

2.3 Threats to validity

The study does not aim at providing a statistically valid view of a certain population of companies, as intended with general surveys [6]. The research questions are focused on existence and not on frequencies of responses. Hence, we consider the survey having more character of multiple case studies on a certain aspect of

Question	Item	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
What is regression testing?	1. Repetitive tests	0	3	3	14	11
	2. Retest of functionality	0	0	1	8	22
	3. Reexecution of testcases	0	1	7	12	11
	4. Same as system testing	11	12	4	2	1
Why is regression testing applied?	5. To assess if the system has the desired properties	1	12	6	8	5
	6. To find defects	0	4	4	15	9
	7. To ensure that nothing has been affected or destroyed	0	0	0	2	30
	8. To guide further priorities in the project	1	9	9	11	0
What kinds of changes generate regression testing?	9. New versions	0	0	3	11	18
	10. New configurations	0	1	7	11	12
	11. Fixes	0	0	6	9	16
	12. Changed solutions	0	1	3	8	18
	13. New hardware	1	5	5	8	11
	14. New platforms	1	3	4	6	17
	15. New designs	0	1	7	9	15
	16. New interfaces	0	2	4	10	15
	17. RT is applied regardless of changes	3	12	6	5	5
At which levels are RT carried out?	18. Single components	3	4	7	11	4
	19. Single modules	1	3	5	15	5
	20. Whole system	0	0	1	9	22
When in the development process is RT applied?	21. As early as possible	3	6	6	7	7
	22. Continuously during the whole process	1	4	5	11	9
	23. At the end	3	1	3	9	16
	24. Daily	8	11	6	2	1
	25. At each software integration	3	4	9	9	5
	26. At each milestone	1	6	6	9	6
	27. Before each release	0	0	1	9	22
	28. As often as we have resources	6	9	5	6	3
What determines the amount and frequency of RT?	29. The assessed risk	0	0	3	14	13
	30. The amount of new functionality	0	2	2	15	12
	31. The amount of fixes	0	2	3	16	9
	32. The amount of available resources	4	6	6	8	6
Which tests are used in regression testing?	33. A selection of developer's tests	5	10	6	6	2
	34. A selection of tester's tests	0	2	1	21	8
	35. A selection from a specific regression test suite	0	4	1	8	18
	36. New test cases are designed	1	9	8	12	2
How are regression test cases selected?	37. The same tests are run each time	1	4	13	8	6
	38. Selection depends on the situation	0	4	6	13	9
	39. We do a complete retest each time	2	8	12	3	6
	40. We do a complete retest of safety critical parts	0	2	9	10	8
	41. Test cases on changes and possible side effects	1	2	5	14	10
	42. A selection is made ad hoc	9	12	7	3	0
	43. Run as many as possible from a prioritized list	4	9	7	6	4
	44. We focus on functional test cases	0	2	8	14	8
	45. We execute smoke test	1	4	11	10	5

Figure 1: Number of responses for each questionnaire alternative on regression test practices

Question	Manual		Equal		Automated
46. How do you execute regression tests?	8	2	15	4	3

Figure 2: Number of responses for each questionnaire alternative on automated vs. manual regression testing

Question	Item	Very dissatisfied	Dissatisfied	Neutral	Satisfied	Very satisfied
How satisfied are you with following in your organization?	47. Processes/practices for change impact analysis	4	7	9	10	1
	48. Assessment of the extent of test coverage	2	6	12	8	2
	49. Assessment of the amount of required tests	1	4	11	14	1
	50. Prioritization of test cases wrt product risks	1	3	9	16	3
	51. Prioritization of test cases wrt fault detection ability	0	4	13	10	2
	52. Time to design good test cases	4	6	9	10	1
	53. Methods/tool support to design good test cases	3	12	9	4	4
	54. Assessment of cost/benefit of automating RT	3	11	12	2	4
	55. Time to RT	3	12	7	7	2
	56. Balance between manual and automated RT	3	14	4	4	4
	57. Execution of automated RT	2	12	8	2	3
	58. Environment for automated RT	2	14	7	3	2
	59. Execution of manual RT	0	2	9	19	0
	60. RT in real target environment	0	4	8	15	4
	61. RT in simulated target environment	0	5	9	13	3
	62. RT of GUI	2	5	7	15	2
	63. RT of data base applications	1	4	10	13	0
	64. RT of third party products	1	5	15	6	0
	65. Consistency of verdict reporting	0	3	14	11	1
	66. Time to analyze results	1	3	15	12	1
	67. Processes/practices for analyzing results	1	5	14	10	1
	68. Presentation of results from automated tests	2	8	6	5	4
	69. Maintenance of tests in case of changes in products	3	7	13	8	1
	70. Methods/tool for traceability between TC and reqs	6	9	6	7	3
	71. Minimization of redundant tests (wrt test coverage)	2	10	8	11	0
	72. Coordination between designers and testers	1	1	8	21	1
	73. Minimization of dependencies in the system	1	12	12	6	0
	74. Modularization of the system	0	9	15	7	0
	75. Testability issues in design guidelines	2	9	14	5	0

Figure 3: Number of responses for each questionnaire alternative on satisfaction with regression test practices

Table 2: Respondents to the questionnaire. Number of developers in the surveyed company: extra small is 1, small is 2 – 19, medium is 20 – 99, and large 100 – 999

Company	Size	Domain
Me	Small	Automation
Te	Medium	Automation
V	Large	Automotive
Tc	Small	Business intelligence
Ql	Medium	Business intelligence
Ti	Medium	Business intelligence
C	Large	Consultant
Ha	Large	Consultant
H	Large	Consultant
H	Large	Consultant
Q	Medium	Consultant
R	Small	Consultant
K	Medium	Consultant
Si	Large	Consultant
So	Large	Consultant
T	Small	Consultant
Tp	Medium	Consultant
Eu	Medium	Finance
Sk	Large	Finance
A	Medium	Finance
U	Medium	Information systems
Sm	Medium	Information systems
W	Small	Information systems
B	Large	Information systems
L	Large	Insurance
Mu	Large	Insurance
Ma	Large	Medical devices
E	Large	Telecom
Hi	Medium	Telecom
M	Medium	Telecom
S	Large	Telecom
S	Large	Telecom

several cases and consequently we discuss threats to validity from a case study perspective [15].

Construct validity concerns the underlying constructs of the research, i.e. terms and concepts under study. We mitigated construct validity threats by having the first question of the focus group related to terminology and concepts. Thereby, we ensured a common understanding for the rest of the group meeting. In the survey, however, the terms may be interpreted differently and this is out of control of the researchers.

Internal validity relates to identification of casual relationships. We do not study any casual relationships in the study, and we just briefly touch upon correlations between factors. Patterns in the data that might indicate correlations are interpreted conservatively in order not to over interpret the data.

External validity relates to generalization from the findings. We do not attempt to generalize in a statistical sense; any generalization possible is analytical generalization [15]. In order to help such generalization, we report characteristics of the focus group members and questionnaire respondents in Tables 1 and 2.

3 Analysis of the results

The focus group and survey results were analyzed using the Zachman framework, which originally was presented for analysis of information systems architectures [18]. The framework has six categories, *what, how, where, who, when* and *why*, although these terms were not originally used. For each category, questions are defined and tailored to the domain under investigation. Originally intended for IS development, Zachman proposed that it might be used for developing new approaches to system development [18]. We use it similar to Runeson [12], i.e. to structure the outcome of the focus group meetings and to define the validation questionnaire, although we primarily focus on *what, how* and *when*.

An overview of the questionnaire results is shown in Figures 1, 2 and 3. Questions are referred to in the text as [Qx] for question *x*. The analysis is then presented according to the framework questions and identified strengths and weaknesses in subsections 3.1 to 3.4.

3.1 What?

There is good agreement in the focus group and among the survey respondents regarding what regression testing is. Regression testing involves repetitive tests and aims to verify that previously working software still works after changes to other parts. Focus can be either re-execution of test cases or retest of functionality.

As for testing in general the goal of the regression testing may differ between different organizations or parts of an organization. The goal may be either to find defects or to obtain a measure of its quality. Regression testing shall ensure that nothing has been affected or destroyed, and give an answer to whether the software has achieved the desired functionality, quality and stability etc. In the focus group discussion, an additional goal of regression testing was mentioned as well; to obtain a guide for further priorities in the project. Regression testing offers a menu of what can be prioritized in the project, such as bug fixes. This additional goal was only confirmed to some extent by 35% of the respondents [Q8].

Different kinds of changes to the system generate regression testing. Mentioned in the focus group discussion and confirmed by the majority of the respondents were: new versions, new configurations, fixes, changed solutions, new hardware, new platforms, new designs and new interfaces [Q9-16]. One third of the respondents, mostly small and medium sized organizations, indicated that regression testing is applied regardless of changes, while in larger organizations, regression testing is tighter connected to changes [Q17]. The amount and frequency of regression testing is determined by the assessed risk, the amount of new functionality, the amount of fixes and the amount of available resources. The first three factors are confirmed by the majority of the respondents [Q29-31] while the agreement on the dependency on resources availability varies to a greater extent among the respondents [Q32].

3.2 When?

Regression testing is carried out at different levels (e.g. module level, component level and system level [Q18-20]) and at different stages of the development process. From focus group discussions it was found that some organizations regression test as early as possible while other regression test as late as possible in the process, and some claimed that regression testing is continuously carried out throughout the whole development process. The purpose may be slightly different for the three options; early regression test to enable early detection of defects, and late regression testing for certification or type approval purposes.

How often regression testing is carried out differed as well; some organizations regression test daily while others regression test at each software integration, at each milestone, or before releases [Q24-26]. In some cases the availability of resources is determinant. Among the questionnaire responses, there were large variations on how often regression testing is applied. The most common approach is to regression test before releases (indicated by 95% of the respondents) [Q27]. Only 10% of the respondents regression test daily [Q24].

3.3 How?

From the focus group discussions it was identified that tests used for regression testing may be a selection of developer's tests, a selection of tester's tests, a selection of tests from a specific regression test suite, or new test cases are designed. According to questionnaire responses, the most common is to reuse test cases designed by testers. Strategies for regression test selection mentioned in the focus group were: complete retest, combine static and dynamic selection, complete retest of safety critical parts, select test cases concentrating on changes and possible side effects, ad-hoc selection, smoke test, prioritize and run as many as possible, and focus on functional test cases. Questionnaire results confirm that it is common to run a set of specified regression test cases every time, together with a set of situation dependent test cases. Ad-hoc selection seems not to be a common approach; only 10% of the respondents indicate that approach [Q42]. 70% of the respondents confirm the focus on functional test cases [Q44] and 50% confirm the usage of smoke tests [Q45].

A project may include several different regression testing activities. Both manual and automatic regression testing are applied. 50% of the respondents indicate an equal amount of manual and automatic regression testing while 30% perform regression testing exclusively manually [Q46].

3.4 Weaknesses and strengths

The focus group had an open discussion about both weaknesses and strengths in their regression testing practices, and it showed that in several cases representatives from one organization had solution proposals where others had problems. Some problems were common to most of the participants (e.g. lack of time and resources to regression test and insufficient tool support) while others were more specific. The outcome of the discussion was a list of 29 possible problem areas which were validated in the questionnaire.

Test case selection. Several problems related to test case selection were discussed in the focus group. It was mentioned that it is hard to assess the impact of changes on existing code and to make a good selection. It is hard to prioritize test cases with respect to product risks and fault detection ability, and to be confident in not missing safety critical faults. Determining the required amount of tests was also considered a problem, and it is hard to assess the test coverage.

Participants wished for a regression test suite with standard test cases and for regression testing guidelines at different stages of a project with respect to quality aspects. Some participants were satisfied with their impact analysis and with their test management systems. As a response to the test selection problem, exploratory testing was recommended and also to have a static test set used for each release.

No specific test selection technique was referred to, such as the ones reviewed by Engström *et al.* [4].

The results from the questionnaire responses are in this respect not conclusive. The responses are divided evenly across the whole spectrum, with a slight shift towards satisfaction. However, in terms of processes for impact analysis and assessment of test coverage the challenges identified in the focus group were confirmed by a third of the respondents even though as many were satisfied. [Q47-51].

Test case design. Lack of time and resources for regression testing was a recurring complaint in the discussions. So also in the case for test case design. Among respondents to the survey were as many satisfied as dissatisfied in this matter [Q52]. One proposal mentioned in the focus group was to focus on test driven development and thus make developers take test responsibility, hence building test automation into the development process, which may be reused for regression testing purposes as well.

Automated and manual regression testing. Automating regression testing causes problems and manual testing is time and resource consuming. Both problems and proposals were discussed in the focus group. Within the focus group, participants were satisfied and dissatisfied with automation as well as with their manual testing. Most participants wanted a better balance between automated and manual testing and support in determining cost benefit of automating regression testing.

It is not only costs for implementing the automated tests that need to be considered, but also costs for maintaining the test suites and in many cases manual analysis of results. It was proposed to define interfaces for automation below the user interface level in order to avoid frequent changes of the test scripts, due to user interface changes. Use of manual testing was recommended for testing of user experience and for exploratory testing.

The problems of automation was confirmed by questionnaire responses. 60% of the respondents were dissatisfied with the balance between manual and automated regression testing [Q56], the assessment of cost/benefit, execution of automated regression tests as well as the environment for automated regression testing. In contrast, as many were satisfied with their manual testing, 60% [Q59].

Regression testing problem areas. Specific problem areas for regression testing, mentioned in the discussion forum were: regression tests in real target environment and in simulated target environment, regression testing of third party products and of GUI's. For each problem mentioned, were among the participants both those who had problems and those who were satisfied with their solutions. None of the problem areas was confirmed by a majority of negative answers in the questionnaire even though between 10-25% were dissatisfied in each case [Q60-

64]. As testing of databases is subject to regression testing research, this area was added to the questionnaire, although not mentioned in the focus group.

Test results. Several of the participants in the focus group were unsatisfied with how test results were presented and analyzed. In many cases verdict reporting is inconsistent and often there is no time to do a thorough analysis. Some participants said that their reporting of results and analysis works well and gave examples of good factors, such as having an independent quality department and having software quality attributes connected to each test case, which is good not only for reporting results but also for prioritization and selection of test cases.

The questionnaire responses were generally neutral regarding consistency of verdict reporting and processes and practices for analyzing results, but agreed that practices for presentation of results from automated tests were not good enough [Q68].

Test suite maintenance. The focus group named maintenance of test suites and test cases as a problem. Participants stated that much of the regression testing is redundant with respect to test coverage and that there is a lack of traceability from tests to requirements. Some of the participants were satisfied with their tools and processes for traceability and claimed that they are good at maintenance of test cases in case of changes in the product. A recommendation was to have independent review teams reviewing the test protocols.

Questionnaire responses confirmed the lack of good tools for documenting traceability between test cases and requirements but otherwise the variation in the responses to the questions regarding maintenance was great [Q69-71].

Testability. An issue brought up in the focus group were the amount of dependencies in the software and its relation to testability. Participants expressed a wish for a test friendly design where the structure enables a simple delimitation of relevant tests. There is a need for design guidelines considering testability, modularization of the software and clearer dependencies in order to make it easier to set test scopes.

Questionnaire responses indicate satisfaction with coordination/communication between designers and testers [Q72] and neutrality to modularization of the system [Q74]. Further they confirmed the need for minimization of dependencies in the system [Q73] as well as for testability issues in design guidelines [Q75].

Test planning. Finally some needs and recommendations regarding the test planning was given. Again a cost model was asked for: *It would be nice to have a cost model for environments and technical infrastructure covering; automated testing, test data, test rigs, unit tests, functional tests, performance tests, target/simulator and test coverage.*

Everyone in the focus group agreed that it is better to test continuously than in large batches. A rule of thumb is to plan for as much test time as development time even when the project is delayed. It is also good to have a process with a flexible scope for weekly regression tests, e.g. core automated scope, user scenarios, main regression scope, dynamic scope, dynamic exploratory scope etc. In order to broaden the coverage, it was proposed to vary the test focus between different test rounds.

4 Conclusions

Regression testing increases in software projects as software becomes more and more complex with increasing emphasis on systematic reuse and shorter development cycles. Many of the challenges, highlighted in the study, are *not* specific to regression testing but are general to all testing. However, they have a significant impact on how effective the regression testing becomes. Questions involving automated testing is of course particularly important for regression testing, as the same tests are repeated many times. Similarly, a test-friendly design is of great importance when one wants to do a selective retesting. Literature on regression testing tends to focus on the selection of test cases based on changes in the code, but for practitioners it does not seem to be the most important issue.

Regression testing definitions (RQ1) are very much the same across all surveyed companies and in line with formal definitions [8] although the regression testing practices differ. Regression testing is applied differently in different organizations, at different stages of a project, at different levels and with varying frequency. Regression testing is not an isolated one-off activity, but rather an activity of varying scope and preconditions, strongly dependent on the context in which it is applied. In most development organizations, regression testing is applied continuously and at several levels with varying goals. This further underlines the need for industrial evaluations of regression testing strategies, where context information is clearly reported, as was previously noted [4].

Regression testing challenges (RQ2) relate to test case selection, trade-offs between automated and manual testing and design for testability. Issues related to test automation are:

- Assessment of cost/benefit of test automation
- Environment for automated testing and the presentation of test results.

Design issues affect regression testing since there is a strong relation between the effort needed for regression testing and the software design. Design for testability, including modularization with well defined and observable interfaces, helps

verifying modules and their impact on the system. This could be addressed by including testability in design guidelines. Except for the design issues, coordination and communication between designers and testers work well.

Good practices (RQ3) were also reported on:

- Run automated daily tests on module level.
- Focus automation below user interface.
- Visualize progress monitoring.

These practices are not specific to regression testing. The latter item is not specific testing at all, but is a management practice that becomes critical to regression testing as it constitutes a key part of the development project progress. This indicates that regression testing should not be addressed nor researched in isolation; rather it should be an important aspect of software testing practice and research to take into account.

Acknowledgment

The authors would like to thank Per Beremark for moderating the focus group meeting and to all participants in the focus group and questionnaire. The work is partly funded by The Swedish Governmental Agency for Innovation Systems (VINNOVA) in the UPPREPA project under grant 2005-02483, and partly by the Swedish Research Council under grant 622-2004-552 for a senior researcher position in software engineering.

References

- [1] Online survey software | SurveyGizmo - affordable enterprise survey software. <http://www.surveygizmo.com>, December 2009.
- [2] Adnan Causevic, Daniel Sundmark, and Sasikumar Punnekkat. An industrial survey on contemporary aspects of software testing. In *Proceedings of the 3rd International Conference on Software Testing Verification and Validation*, pages 393–401, 2010.
- [3] Pavan Kumar Chittimalli and Mary Jean Harrold. Recomputing coverage information to assist regression testing. *IEEE Transactions on Software Engineering*, 35(4):452–469, 2009.
- [4] Emelie Engström, Per Runeson, and Mats Skoglund. A systematic review on regression test selection techniques. *Information and Software Technology*, 52(1):14–30, January 2010.
- [5] 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 (ICST'10)*, pages 75–78, 2010.
- [6] Arlene Flink. *The survey handbook*. SAGE Publications, 2nd edition, 2003.
- [7] Mats Grindal, Jeff Offutt, and Jonas Mellin. On the testing maturity of software producing organizations. In *Testing: Academia & Industry Conference-Practice And Research Techniques (TAIC/PART)*, 2006.
- [8] IEEE. Standard for software test documentation. Technical Report 829-1983, Revision, 1998.
- [9] Akira K. Onoma, Wei-Tek Tsai, Mustafa Poonawala, and Hiroshi Suganuma. Regression testing in an industrial environment. *Communications of the ACM*, 41(5):81–86, May 1998.
- [10] Colin Robson. *Real World Research*. Blackwell Publishing, 2nd edition, 2002.
- [11] John Rooksby, Mark Rouncefield, and Ian Sommerville. Testing in the wild: The social and organisational dimensions of real world practice. *Computer Supported Cooperative Work (CSCW)*, 18(5):559–580, 2009.
- [12] Per Runeson. A survey of unit testing practices. *IEEE Software*, 23(4):22–29, 2006.

-
- [13] Per Runeson, Carina Andersson, and Martin Höst. Test processes in software product evolution - a qualitative survey on the state of practice. *Journal of Software Maintenance and Evolution: Research and Practice*, 15:41–59, 2003.
 - [14] Per Runeson, Per Beremark, Bengt Larsson, and Eric Lundh. SPIN-syd—a non-profit exchange network. In *1st International Workshop on Software Engineering Networking Experiences*, 2006.
 - [15] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, April 2009.
 - [16] Mats Skoglund and Per Runeson. A case study of the class firewall regression test selection technique on a large scale distributed software system. In *International Symposium on Empirical Software Engineering*, pages 72–81, 2005.
 - [17] Lee White and Brian Robinson. Industrial real-time regression testing and analysis using firewalls. In *Proceedings of the 20th IEEE International Conference on Software Maintenance (ICSM '04)*, pages 18–27, 2004.
 - [18] John A. Zachman. A framework for information systems architecture. *IBM Systems Journal*, 26(3):276–293, 1987.

INDIRECT EFFECTS IN EVIDENTIAL ASSESSMENT: A CASE STUDY ON REGRESSION TEST TECHNOLOGY ADOPTION

Abstract

Background: There is a need for efficient regression testing in most software development organizations. Often the proposed solutions involve automation of some sort. However, despite this being a well researched area, research results are rarely applied in industrial practice. *Aim:* In this paper we aim to bridge the gap between research and practice by providing examples of how evidence-based regression testing approaches can be adopted in industry. We also discuss challenges for the research community. *Method:* An industrial case study was carried out to evaluate the possibility to improve regression testing at Sony Ericsson Mobile Communications. In this paper we analyse the procedure undertaken based on frameworks from the evidence based software engineering, EBSE, paradigm (with a focus on the evidence) and automation literature (with a focus on the practical effects of the automation changes). *Results:* Our results pinpoint the need for systematic approaches when introducing new tool into the regression testing process. Practitioners and researchers need congruent guidelines supporting the appraisal of both the evidence base and the pragmatic effects, both direct but also indirect, of the changes. This is illustrated by the introduction of the automation perspective, where the indirect effects are often hard to evaluate, but is of general importance. The lack of generalizable results in regression testing literature prevents selection of techniques based on empirical evidence.

Emelie Engström, Robert Feldt and Rickard Torkar

The 2nd International Workshop on Evidential assessment of Software Technologies (EAST'12), Sept 2012

1 Introduction

The cost for testing software increases with the increasing size and complexity of software systems. At the same time more efficient software development strategies have emerged which demand more frequent testing, e.g. agile strategies with continuous integration of software updates, or testing of a broader scope of products, e.g. software product line development [2, 18]. Thus the *share* of testing costs of the total development cost increases as well and testing easily becomes a schedule or cost bottleneck or it becomes inadequate (in worst case both). Regression testing is conducted in order to verify that changes to a system has not negatively impacted on previously functioning software. Thus efficient regression testing strategies are especially important in organizations where software is frequently updated or when a range of different products are derived from the same software base.

However, there is a gap between research and practice of regression testing. Even though several systematic approaches for both prioritization and selection of regression test cases have been proposed and evaluated in literature [7, 19], they are not widely used in industry [4]. In many cases the methods and techniques do not scale while in other cases industry practice is not mature enough to implement them.

Evidence based software engineering, EBSE, is a paradigm aiming to support adoption of new software technology and methods based on sound, empirical evidence. The core tool of EBSE is systematic literature reviews and the focus is on the identification and appraisal of evidence. However, EBSE have been shown to be very challenging for non-researchers and rigorous and relevant evidence may not exist to answer their questions [13, 14].

In this paper we aim to bridge the gap between research and practice by providing examples of how EBSE can be applied in industry and pinpointing challenges for the research community based on experiences from a case study [6] carried out at Sony Ericsson Mobile communications, SEMC. We describe and analyze the procedures undertaken to find a relevant regression testing technique and to evaluate the benefits of introducing it into the testing process.

Introducing a new tool into a process have both direct and indirect effects and it is not enough to focus on direct evidence isolated to a technique but the context and the practical effects of the changes also need to be considered. Thus we combine guidelines on EBSE [3] by Dybå et al. with guidelines on automation with respect to human-computer interaction [11] by Parasuraman et al. as a framework for our analysis. A recent example of automation analysis in technology adoption in

software engineering is given by Borg [1]. This perspective is essential also when considering evidence for the efficacy of regression testing since it will involve automation of some sort; we must therefore better understand the practical effects of introducing automation, in general. For each of the five steps of EBSE, the evaluation and decision tasks of the automation procedure are discussed and embodied with examples from our case study, followed by a discussion about current challenges and advice for practitioners and researchers.

The paper is organized as follows: Section 2 describes the synthesis of the EBSE and automation model and Section 3 reports the case study and contains the analysis. Both these sections are organized according to the five different steps of EBSE. In section 4 we discuss the lessons learned about the methodological strengths and limitations of evidence-based practice and in Section 5 we conclude the paper with advice for practitioners and researchers.

2 The regression test automation procedure

Dybå et al. [3] propose a systematic EBSE approach for practitioners which involves five steps to support and improve decisions on technology adoption. Similarly, Parasuraman et al. [11] propose a procedure involving four steps of evaluation and decision tasks that helps answer questions about what should be automated and to what level. The procedure has primary evaluation criteria based on a model of the main types of human information processing but also includes secondary evaluation criteria based on the reliability of the automation and its effects on cost.

Fig 1 illustrates how the models of Parasuraman et al and Dybå et al complement each other in case of introducing automation to the software engineering process. In this section we describe the synthesis of the EBSE and automation approaches in more detail as well as its application on regression test automation. This framework forms the basis for the description and analysis of the case study procedure in Section 3. Below we describe it according to the five steps in the EBSE approach.

2.1 Ask an answerable question.

This step corresponds to the initial steps in the Automation model answering questions on what to automate and what type of automation should take place. Parasuraman suggest a classification of automation types based on four types of human information processing. This leads to four different classes of functions that can be automated: information acquisition, information analysis, decision and action selection, and action implementation [11].

In case of test automation, the type of automation depends on the *scope* of the testing. Testing software typically involve several test activities which all may be

EBSE-approach	Automation-approach
<i>Focus on research answering the question which technique to use for the automation</i>	<i>Focus on practice answering the question what should be automated</i>
1. Ask an answerable question	1. Identify type of automation
2. Find the best evidence	
3. Critically appraise the evidence	
4. Apply the evidence	2. Identify level of automation
	3. Apply primary evaluation criteria
5. Evaluate performance	4. Apply secondary evaluation criteria

Figure 1: Overview of the EBSE approach described by Dybå et al. [3] and the Automation approach described by Parasuraman et al. [11]. The two models complement each other in their different focus. The Automation model distinguishes between different types and levels of automation and suggests primary and secondary evaluation criteria to be applied iteratively while adjusting the decisions on type and level of automation. The EBSE approach is more general, i.e. not technique-specific.

automated in different ways with different goals: test planning, test case design, test execution and analysis of test results. Automation may seek to replace humans performing repetitive and simple tasks e.g. executing test cases or have a goal to accomplish something beyond human capabilities e.g. decision support. Regression testing research focus on the latter.

The level of testing under consideration is another factor of the automation scope. Goals and strategies for automation may vary with level of test [4]. Test planning at unit level could for example be based on code coverage analyses while at the integration testing level combinatorial strategies could be more relevant. Automation at system level is more complex and a more thorough cost benefit analysis could be a good idea.

Many challenges in regression testing are not specific for regression testing but improvements may be achieved through improvements of other tasks e.g. automation of test executions or design for testability [4]. Hence, it is important to specify the *effect targets* of the intended automation and ensure that they are aligned with the scope of the automation.

A good understanding of the context helps to identify, relevant effect targets and a proper scope for the automation. Even though there is a common understanding of what regression testing is and why it is applied in industry, the variation in practices is large. Regression testing is applied differently in different

organizations, at different stages of a project, at different levels and with varying frequency [4]. Furthermore regression testing is not an isolated activity but is strongly dependent on the context in which it is applied.

2.2 Find the best evidence

The selection of technique to implement should ideally be based on research, and guided by relevant classifications of empirical evidence, typically systematic literature reviews, which could be mapped to a similarly relevant description of the current context, scope and effect target. To determine which regression testing technique is appropriate for a certain situation, several factors need to be considered such as which input does the technique require?, on what level of abstraction is the analysis made?, what are the claims of a technique, and what empirical support is there for these claims? [7]. This step has no counterpart in the Parasuraman model since they assume a different situation in which a specific type of automation has already been selected or is being evaluated. In the EBSE model one should consider the breadth of available evidence and techniques.

Several literature reviews on regression testing have been published lately. A thorough overview of different regression testing strategies in literature is given by Yoo and Harman [19] and it provides a good starting point for the search. Here the different areas of minimization, prioritization and selection are explained and investigated. A classification scheme and classification of regression test selection techniques is provided by Engstrom et al. [7].

2.3 Critically appraise the evidence

This is also a step which is specific for the EBSE approach. EBSE recommend practitioners to make use of available summaries of evidence, e.g. systematic literature reviews and systematic maps. In cases where such summaries are not available practitioners may be guided by checklists [3] in appraising (often a smaller set of) published studies.

The current evidence base on regression test selection does not offer much support for selecting the best automation technique. Although several empirical evaluations of regression testing techniques have been made lately, the evidence base is incomplete and sometimes even contradictory. Regression testing techniques are context dependent and in many cases evaluations are made only in one specific context. It is rather the different implementations of the techniques than the high level concepts that are evaluated. Moreover, often only one aspect is evaluated for a technique e.g. either the level of cost reduction or the level of fault detection [7]. Indirect effects of introducing this type of automation are rarely discussed.

2.4 Apply the evidence

In this step the evidence should be integrated with the knowledge about the concrete situation and the pragmatic effects of the changes should be considered. Automation changes may have both direct and indirect effects. This step concerns the indirect effects of the automation change, for example cognitive effects, effects on related processes (e.g. requirements engineering and design) and long term effects, and corresponds with the identification of level of automation and application of primary evaluation criteria in the Parasuraman model. Sheridan and Verplank define 10 different levels of automation [17] based on the amount of user interaction required. Although they are not defined with a focus on software testing they are applicable in this context.

The primary evaluation criteria according to Parasuraman et al focus on the consequences for the human performance after the automation has been implemented. For example, Parasuraman states that ‘evidence suggests that well-designed information automation can change human operator mental workload to a level that is appropriate for the system tasks to be performed’ [11]. An example can be the listing of test cases in a prioritized list to help testers make the right decision. In addition to mental workload, Parasuraman mentions situation awareness, complacency and skill degradation as primary evaluation criteria to be taken into consideration.

To help better understand the changes that an automation leads to we have found it useful to establish pre- and post-automation task flows. A detailed such analyses could involve the use of value stream maps [10] but we have found even simpler identification procedures to have value. Based on the description of the current testing task flow, and an initial decision on the technique to use for the automation (or at least the type of task to be automated a la Parasuraman), changes between current practice and improved automated practice can be identified. It is useful to consider different types of changes such as: *changed* tasks, *added* tasks and *removed* tasks. It is important not to miss any added tasks such as creation or collection of inputs that the automated process or tool requires, maintaining the automation in case of context changes or managing and processing the output from the automated process.

2.5 Evaluate performance

This step concerns the direct effects of the automation and may be evaluated against the effect targets identified in the first step. For this step the EBSE and automation guidelines are consistent. The selected technique should be evaluated within the context where it is to be applied. Results from this evaluation should be documented and may be reused in future automation projects either within the organization or as a contribution to the general knowledge base.

3 The case study

The case study was carried out in four steps [6] starting with A) exploratory semi-structured interviews to better understand the context, effect targets and scope of the automation. B) Select and implement a suitable method. The methods were C) quantitatively evaluated with respect to their fault detection efficiency, and finally D) the testers' opinions about the implemented methods were collected and analyzed. Results from this case study has previously been reported by Engstrom et al. [6]. In this section we describe our application of the regression test automation procedure in Section 2 and discuss current challenges and advices for practitioners and researches.

3.1 Asking an answerable question.

This step involves describing the context and identifying the effect targets of the regression test automation.

Describing the context – To gain more insights into the problem we carried out interviews with a number of key persons and used a framework proposed by Petersen and Wohlin [12] to structure the context information. The context description served several purposes, identification of context constraints relevant for the selection of technique, support in the evaluation of indirect effects of changes as well as support in the communication of the results of the evaluation of direct effects through the enabling of analytical generalization. Details about the context description are reported in [6]

Identifying the effect target – One of the problems with the current method in our case study was its dependence on experienced testers with knowledge about the system and the test cases. There was a risk that the selected test suite was either too extensive or too narrow; a tester with lack of experience in the area could have trouble estimating the required time and resources. Moreover, the selected test suite could be inefficient and misleading. Even experienced testers could select inefficient test suites since test cases were selected in a routinely manner by just selecting the same test cases for every regression test session. Since the selection was based on judgment, there was no evidence that it was the most efficient test suite. Hence, the following were the expected benefits of a tool supported selection procedure:

- increased transparency
- improved cost estimation
- increased test efficiency

- increased confidence

The scope and effect targets identified in our case study correlate with general needs in industry [4] and is partly inline with the scope of most regression test selection and prioritization techniques in literature [19]. Similar problems were identified in a survey on regression testing practices [4]. Participants found it hard to assess the impact of changes on existing code and to make good selections, to prioritize test cases with respect to product risks and fault detection ability, and to be confident in not missing safety critical faults. Determining the required amount of tests was also considered a problem as well as to assess the test coverage.

3.2 Finding the best evidence

The selected scope of the automation directed the search towards the regression testing literature, and the list of effect targets and context constraints influenced the selection of automation technique. The hypothesis that history-based test case prioritization could improve the current situation was a starting point for the case study which to some extent made the search and selection of technique biased. Only history-based regression testing techniques which did not require source code access were considered.

Most of the proposed techniques in literature are code based (source code, intermediate code or binary code) and based on analysis at a low level of abstraction (e.g. statements). However there are examples of techniques based on analysis at higher abstraction levels as well as on other types of input. Some techniques are based on a certain type of specifications or models. There are also some recent examples of both selection and prioritization techniques based on project data, such as failure reports or execution history of a test case. One group of techniques is claimed to be safe, meaning they do not exclude any test case that have a possibility to execute parts of the system that may have been affected by a change. This property is only relevant for selection, but might be important for certain types of systems (e.g. safety critical) [7].

3.3 Appraising the evidence

In our case the selection of technique was not guided by empirical evidence on the performance of the technique. Instead the automation choice was guided by the reports of the nature of the proposed techniques. The technique proposed by Fazlalizadeh et al. [8] was selected because it was based on historic project data, could be implemented without access to source code and allowed for multiple prioritization criteria.

3.4 Applying the evidence

Our case study spans one iteration over the steps described in 2, identification of level of automation, identification of changes and evaluation of the effect of changes.

3.4.1 Identification of level of automation

In our case the automation level would be 4: “*Computer offers a restricted set of alternatives and suggests one, but human still makes and implements final decision*” according to Sheridan and Verplank [17]. Since the scope of the automation in the case study is pure decision support and not execution of tests, it corresponds to a low level of automation. On the other hand there are differences in levels within this scope that could be interesting to pinpoint. Instead of calculating total priority values by combining several prioritization criterion, priority values could be calculated for each criteria leaving the weighting of importance to the human which would be an example of automation at even lower level but still not zero.

3.4.2 Identification of changes

The scope of the automation in our case study involved the test planning and selection of test suites for regression testing. No common procedure for how this is done existed, but different testers developed their own strategies. Thus the task flow directly affected by the automation cannot be described in more detail. This situation seems to be common in industry. [4]

Changed, added and removed tasks in the case study may be described as follows:

Changed tasks – The suggested automation would change the selection task from being an ad-hoc task to involve a more systematic procedure. Decisions on focus, scope and constraints of a test session are needed as input to the tool and thus have to be explicit.

Added tasks – New tasks relate to the use of the tool. There is an initial learning effort for using the tool. It also has to be maintained and evolve as the prerequisites change. The outcome of the tool (a selected test suite) needs to be reviewed before test execution. In this case the tool was a plug in to the supporting tool already in use at the company and it collected data from the existing test data base. No input data of any other format is required and thus no such extra tasks (development and maintenance of new artifacts) are added.

Removed tasks – The selection and documentation of test scope are achieved by the tool instead of manual procedures.

3.4.3 Evaluation of the effect of changes

Parasuraman's list of cognitive effects may be used as a checklist for one type of indirect effects. Another checklist may be created by combining the list of task changes with the previously described elements of the context into a matrix. Each item in the matrix represents a combination of a task change and a context facet to be considered. The indirect effects discussed in this section are identified with the help of both these checklists.

The mental effort for the decisions may increase since in many cases these types of decisions are made ad-hoc in a routinely manner. With a semi-automatic tool testers are forced to perform the selections more systematically and of course to learn and handle a tool. This increased effort might lead to that the tool is not used properly especially since the regression testing activity is so frequent. On the other hand they do not have to browse through large amounts of poorly structured test cases.

There is also *a risk in trusting the tool too much*. Decisions are complex and context dependent and the variation in regression test situations cannot be fully captured by a tool. In our case study the product maturity is low and the software system complex. The large amount of internal and hidden dependencies prevent safe selections at a reasonable cost. Furthermore, in a very changing context, historical data might not be a good predictor of fault detection probability at all. However, the quantitative evaluation in our example case showed increased efficiency in that particular case, indicating a possibility to improve regression testing with the tool. Still the suggested test suite need to be manually reviewed. Only a low level of automation is possible to achieve.

With a low level of automation the *risk of skill degradation and decreasing situation awareness* is minimal. Changing the task from manually selecting each test case to manually review a suggested test suite would probably increase the testers awareness of unexpected dependencies but also decrease their awareness of what is in the test data base and what might be missing. A more transparent regression testing approach eases synchronization of work between teams and enables their cross learning.

A positive effect of automatically generating the test plan in the test management tool instead of manually entering the test cases is *increased consistency in the documentation*. Lack of consistency in test documentation may lead to unnecessary duplication in distributed testing [5]

3.5 Evaluating performance and seek ways to improve it.

The direct effects of the implemented automation were shown to be *increased transparency* and *increased efficiency*. A prototype tool was developed for the purpose of evaluating the automation in the case study. Adaptations to the actual context were inevitable and two different versions of the technique were implemented: one as close to the original proposal as possible and one as close to the

current decision procedure as possible incorporating a number of different prioritization criteria that was assumed to be important by the local experts. These criteria were identified through the initial interviews and included: historical effectiveness, execution history, static priority, age, cost, focus of test session, scope of session and current status of the test case. A more detailed description of these factors are found in [6]

Two quantitative evaluations and one qualitative evaluation were carried out to evaluate the effects of automation (Details about the evaluations are provided by Engstrom et al. [6]) with respect to the effect targets: increased transparency, increased test efficiency and increased confidence. No data regarding the *execution cost* for a test case or group of test cases was available and thus this aspect could not be evaluated.

Increased transparency is achieved with any kind of automation, since no systematic method for regression testing was currently in use. If the use of a systematic method or implemented tool does not decrease test efficiency or confidence, it is considered an improvement of the current situation.

Test efficiency regards the number of faults revealed per executed test case and was evaluated in two ways in the case study: 1) by comparing the efficiency of the execution order (prioritization) of test cases in the suites and 2) by analyzing test suite selections of the same magnitude as corresponding manually selected suites. Results from the quantitative evaluation constitutes strong evidence for the possibility to improve efficiency in the current case but little relevance in terms of possibility to generalize to other contexts.

To reach and measure *confidence* in a method is in itself non-transparent, since it deals with the gut feelings of the testers. To some extent it relates to coverage. If a method can be shown to include all presumed important test cases, the confidence in it is high. However, optimizing a method to include presumed important coverage aspects affect test efficiency negatively, since it adds test cases to the selection without respect to their probability of detecting faults.

4 Discussion

In many cases adoption of software engineering evidence in practice involve some type of automation. When changing the level of automation in a process it is not enough to appraise the direct evidence of a technique since any type of automation also entails indirect effects. In our case study we identified a number of indirect effects: increased mental effort for decisions, risk for over trust in tool, skill degradation and decreased situation awareness. Also, positive indirect effects were identified such as increased situation awareness, increased consistency in test documentation, easier synchronization of work and cross learning between teams. It is not enough to answer the question “Are these evidence valid, have impact and applicable?” it is critical to consider the questions “What are indirect/additional

effects when applying this and what type of evidence is there that negative indirect effects do not outweigh any positive direct effects?”.

Our analysis of the regression test automation procedure show the importance of finding holistic guidelines to support the adoption of evidence, focusing on both evidence and practice. This is central for EBSE to have effect. There is a need for both general guidelines and guidelines for specific sub areas within software engineering. In our cased study we created two checklists one with automation effects to look for, based on experiences from the human-computer interaction and software test automation disciplines [9, 11], and one, pointing at changes in the context, based on the elements of the context description combined with the three types of changes: added, removed and changed tasks.

In addition to the need for methods to support the evaluation of indirect effects, our case study shows the importance of finding methods to match the communication of empirical evidence with guidelines for identifying context constraints and effect targets present in practice. This is typically an area where guidelines need to be both general and sub area specific. To provide support for practitioners in matching their context with the available evidence, frameworks must be defined at an abstraction level low enough to cover the relevant aspects of the particular group of techniques addressing their effect targets. Still the more abstract framework is needed to initially identify the scope and effect targets.

Finally, another critical issue for the success of EBSE is the existence of generalizable results. Artifacts in software engineering research tend to be context dependent, calling for systematically replicated evaluations where context factors are considered. In the case of regression test automation there is a need for evaluations of high level concepts of strategies rather than evaluations of specific implementations of techniques. To support such research it is crucial to identify the relevant context factors for a specific sub-area and define frameworks at a proper level of abstraction.

5 Conclusion

We have analyzed the procedure of finding a proper technique for regression test automation and evaluating the benefits of adopting it. Frameworks for EBSE and automation were combined to provide a more holistic view of the regression test automation procedure. For each of the five EBSE steps we describe the regression test automation procedure and provide examples of how to do for practitioners in similar situations. Advice for practitioners may be summarized regarding the identification of preconditions for the automation, identification of relevant research and evaluation of effects of the automation.

Identification of preconditions – It is important to identify scope and effect targets of automation. Information may be gathered with interviews and struc-

tured according to existing frameworks [12] for context descriptions in software engineering.

Identification of relevant research – The search for relevant techniques should be driven by scope of automation, effect targets and context constraints. Effect targets in our case study correlate with general needs in industry and are partly addressed by research. Literature reviews on regression testing [7, 19] provide overviews of the research and are good starting points for the search. However the lack of generalizable results prevents choices based on empirics.

Evaluation of effects – Automation changes may have both direct and indirect effects. To identify indirect effects changed, added and removed tasks need to be identified and the effects of the changes evaluated. Checklists may guide the analysis. The direct effects may be evaluated against the effect targets. Frameworks for evaluating cost and efficiency of regression testing techniques are available [15, 16]

Research challenges involve 1) finding methods for evaluating indirect pragmatic effects of the automation changes as well as 2) finding methods to match the communication of the empirical evidence with the description of the industrial context where automation is under consideration.

Acknowledgments

We thank Dr. Per Runeson for valuable review comments on this paper. The work was supported by VINNOVA (the Swedish Governmental Agency for Innovation Systems) to SWELL, Swedish research school in Verification and Validation.

References

- [1] Markus Borg. Findability through traceability - a realistic application of candidate trace links? In *Proceedings of the 7th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2012)*, pages 173–181, June 2012.
- [2] Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, Boston, 2001.
- [3] Tore Dybå, Barbara A. Kitchenham, and Magne Jørgensen. Evidence-based software engineering for practitioners. *IEEE Software*, 22(1):58–65, January 2005.
- [4] 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.
- [5] Emelie Engström and Per Runeson. Test overlay in an emerging software product line—an industrial case study. *Information and Software Technology*, 55(3):581–594, March 2013.
- [6] 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 (ICST'11)*, pages 367–376, 2011.
- [7] Emelie Engström, Per Runeson, and Mats Skoglund. A systematic review on regression test selection techniques. *Information and Software Technology*, 52(1):14–30, January 2010.
- [8] Yalda Fazlalizadeh, Alireza Khalilian, Mohammad Abdollahi Azgomi, and Saeed Parsa. Prioritizing test cases for resource constraint environments using historical test case performance data. In *Proceedings of the 2nd IEEE International Conference on Computer Science and Information Technology (ICCSIT 2009)*, pages 190–195, August 2009.
- [9] Mark Fewster and Dorothy Graham. *Software Test Automation*. Addison-Wesley Professional, September 1999.
- [10] Shahid Mujtaba, Robert Feldt, and Kai Petersen. Waste and lead time reduction in a software product customization process with value stream maps. In *Australian Software Engineering Conference*, pages 139–148, 2010.

-
- [11] Raja Parasuraman, Thomas B. Sheridan, and Christopher D. Wickens. A model for types and levels of human interaction with automation. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 30(3):286–297, May 2000.
 - [12] Kai Petersen and Claes Wohlin. Context in industrial software engineering research. In *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM '09)*, pages 401–404, October 2009.
 - [13] Austen Rainer and Sarah Beecham. A follow-up empirical evaluation of evidence based software engineering by undergraduate students. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, 2008.
 - [14] Austen Rainer, Tracy Hall, and Nathan Baddoo. A preliminary empirical investigation of the use of evidence based software engineering by undergraduate students. In *Proceedings of the 10th International Conference on Evaluation and Assessment in Software Engineering*, 2006.
 - [15] Gregg Rothermel and Mary Jean Harrold. Analyzing regression test selection techniques. *IEEE Transactions on Software Engineering*, 22(8):529–551, August 1996.
 - [16] Gregg Rothermel, Roland H. Untch, Chu Chengyun, and Mary Jean Harrold. Test case prioritization: an empirical study. In *Proceedings of the IEEE International Conference on Software Maintenance*, pages 179–188, 1999.
 - [17] Thomas B. Sheridan and William L. Verplank. Human and computer control of undersea teleoperators. Technical report, MIT Man-Machine Laboratory, July 1978.
 - [18] David Talby, Arie Keren, Orit Hazzan, and Yael Dubinsky. Agile software testing in a large-scale project. *IEEE Software*, 23(4):30–37, 2006.
 - [19] Shin Yoo and Mark Harman. Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification and Reliability*, 22(2):67–120, March 2012.

SOFTWARE PRODUCT LINE TESTING – A SYSTEMATIC MAPPING STUDY

Abstract

Context: Software product lines (SPL) are used in industry to achieve more efficient software development. However, the testing side of SPL is underdeveloped. *Objective:* This study aims at surveying existing research on SPL testing in order to identify useful approaches and needs for future research. *Method:* A systematic mapping study is launched to find as much literature as possible, and the 64 papers found are classified with respect to focus, research type and contribution type. *Results:* A majority of the papers are of proposal research types (64 %). System testing is the largest group with respect to research focus (40 %), followed by management (23 %). Method contributions are in majority. *Conclusions:* More validation and evaluation research is needed to provide a better foundation for SPL testing.

Emelie Engström and Per Runeson

Journal of *Information and Software Technology* 53(1):2-13, 2011

1 Introduction

Efficient testing strategies are important for any organization with a large share of their costs in software development. In an organization using software product lines (SPL) it is even more crucial since the share of testing costs increases as the development costs for each product decreases. Testing of a software product line is a complex and costly task since the variety of products derived from the product platform is huge. In addition to the complexity of stand-alone product testing, product line testing also includes the dimension of what should be tested

in the platform and what should be tested in separate products. Early literature on product lines did not spend much attention to testing [7, p278-279], but the issue is brought up after that, and much research effort is spent on a variety of topics related to product line testing. In order to get a picture of existing research we launched a systematic mapping study of product line testing. The aim is to get an overview of existing research in order to find useful results for practical use and to identify needs for future research. We provide a map over the existing research on software product line testing. Overviews of challenges and techniques are included in several earlier papers, as well as a couple of brief reviews. However no extensive mapping study has been reported on earlier.

Systematic mapping is a relatively new research method in software engineering, adapted from other disciplines by Kitchenham [31]. It is an alternative to systematic reviews and could be used if the amount of empirical evidence is too little, or if the topic is too broad, for a systematic review to be feasible. A mapping study is performed at a higher granularity level with the aim to identify research gaps and clusters of evidence in order to direct future research. Some reports on systematic mapping studies are published e.g. on object-oriented software design [2] and on non-functional search-based software testing [1]. Petersen et al. [58] describe how to conduct a systematic mapping study in software engineering. Our study is conducted in accordance with these guidelines. Where applicable, we have used the proposed classification schemes and in addition, we have introduced a scheme specific to our topic. This paper is organized as follows: Section 2 describes how the systematic mapping methodology has been applied. Section 3 summarizes challenges discussed in literature in response to our first research question. In section 4 we compile statistics on the primary studies to investigate the second research question. Section 5 presents the classification schemes used and in section 6 the actual mapping of the studies, according to research questions three and four, is presented together with a brief summary of the research. Finally, discussion and conclusions are provided in sections 7 and 8, respectively.

2 Research method

2.1 Research questions

The goal of this study is to get an overview of existing research on product line testing. The overall goal is defined in four research questions:

- *RQ1 Which challenges for testing software product lines have been identified?* Challenges for SPL testing may be identified in specific surveys, or as a bi-product of other studies. We want to get an overview of the challenges identified to validate the relevance of past and future research.
- *RQ2 In which fora is research on software product line testing published?* There are a few conferences and workshops specifically devoted to SPL.

However, experience from earlier reviews indicates that research may be published in very different fora [15].

- *RQ3 Which topics for testing product lines have been investigated and to what extent?* As SPL is related to many different aspects, e.g. technical, engineering, managerial, we want to see which ones are addressed in previous research, to help identifying needs for complementary research.
- *RQ4 What types of research are represented and to what extent?* Investigations on types of research in software indicate that the use of empirical studies is scarce in software engineering [21]. Better founded approaches are advised to increase the credibility of the research [69] and we want to investigate the status for the specific subfield of SPL testing.

2.2 Systematic mapping

In order to get an overview of the research on SPL testing, a systematic mapping study is carried through. A detailed description on how to conduct systematic mapping studies, and a discussion of differences between systematic mapping and systematic reviews, is presented by Petersen et al. [58]. The mapping process consists of three activities; i) search for relevant publications, ii) definition of a classification scheme, and iii) mapping of publications. In this study, search for publications is done in five steps of which the two last steps validate the search, see Figure 1, using a combination of data base searches and reference based searches [70]. In the first step an initial set of papers was identified through exploratory searches, mainly by following references and links to citing publications, with some previous known publications as the starting point [42,47,50,59,60,73] The result of this activity was 24 publications, which were screened in order to retrieve an overview of the area; frequently discussed challenges, commonly used classifications and important keywords.

The second step consisted in reading introduction sections and related works sections in the initial set of publications and extending the set with referenced publications relevant to this study. Only papers with a clear focus on the testing of a software product line published up to 2008 were included. This resulted in additional 33 publications. In order to avoid redundancy in research contributions and to establish a quality level of included publications we decided however to narrow down the categories of publications after this stage. Non peer reviewed publications; such as technical reports, books and workshop descriptions, in total 23 publications, were excluded from the set of primary studies. Among those is an early technical report by McGregor [42] (cited in 70 % of the publications) which is used to find relevant primary studies, but not included among the primary studies as such. Another result of this step was a summary of challenges in SPL testing identified by the community and a preliminary classification scheme for research contributions.

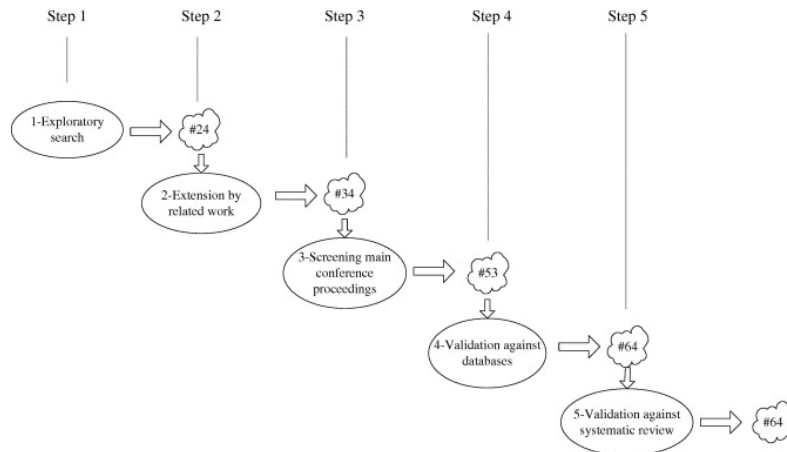


Figure 1: Search for publications on software product line testing

In the third step we screened titles in proceedings from the most frequent publication forum from the previous steps; the workshop on Software Product Line Testing (SPLiT), and from the corresponding main conference; the Software Product Line Conference (SPLC). The number of primary studies is 53 after this step.

The fourth and fifth steps are validating the first three. The fourth step includes automatic searches with Google Scholar and ISI Web of science. The search string was “product” and “line/lines/family/families” and “test/testing” and it was applied only to titles, which has shown to be sufficient in systematic reviews [12]. This search resulted in 177 hits in Google Scholar and 38 hits in ISI Web of science. The search in web of science did not result in any new unique contribution.

Excluded publications were, except for the above mentioned, tool demonstrations, talks, non-english publications, patent applications, editorials, posters, panel summaries, keynotes and papers from industrial conferences. In total 49 publications were relevant for this study according to our selection criteria. This set was compared to our set of 53 papers from step three and 38 papers were common. The differing 11 publications were added to the study. In the fifth step the set of papers was compared to a set of paper included in a systematic review on product line testing by Lamancha et al. [38]. Their study included 23 papers of which 12 passed our criteria on focus and publication type. All of these were already included in our study. Thus we believe that the search for publications is sufficiently extensive and that the set of publications gives a good picture of the state of art in SPL testing research.

A summary of the inclusion and exclusion criteria is:

- Inclusion: Peer reviewed publications with a clear focus on some aspect of software product line testing.

- Exclusion: Publications where either testing focus or software product line focus is lacking. Non-peer reviewed publications.

The answer to RQ1 was retrieved through synthesising the discussions in the initial 24 publications until saturation was reached. Several publications are philosophical with a main purpose to discuss challenges in SPL testing and almost all papers discuss the challenges to some extent in the introductory sections. All challenges mentioned were named and grouped. A summary of the challenges is provided in section 3. Answers to questions RQ2, RQ3 and RQ4 are retrieved through analysing the 64 primary studies. A preliminary classification scheme was established through keywording [58] abstracts and positioning sections. Classifications of the primary studies were conducted by the first author and validated by the second. Disagreements were resolved through discussions or led to refinement of the classification scheme, which in turn led to reclassification and revalidation of previously classified publications. This procedure was repeated until no disagreements remained.

2.3 Threats to validity

Threats to the validity of the mapping study are analyzed according to the following taxonomy: construct validity, reliability, internal validity and external validity.

Construct validity reflects to what extent the phenomenon under study really represents what the researchers have in mind and what is investigated according to the research questions. The terms product lines, software product lines and family/families are rather well established, and hence the terms are sufficiently stable to use as search strings. Similarly for testing, we consider this being well established. Another aspect of the construct validity is assurance that we actually find all papers on the selected topic. We have searched broadly in general publication databases which index most well reputed publication fora. The long list of different publication fora indicates the width of the searching is enough. The snowball sampling procedure has been shown to work well in searching with a specific technical focus [70]. We also validated our searches against another review, and found this review covering all papers in that review.

Reliability focuses on whether the data are collected and the analysis is conducted in a way that it can be repeated by other researchers with the same results. We defined search terms and applied procedures, which may be replicated by others. The non-determinism of one of the databases (Google scholar) is compensated by also using a more transparent database (ISI Web of Science). Since this is a mapping study, and no systematic review, the inclusion/exclusion criteria are only related to whether the topic of SPL testing is present in the paper or not. The classification is another source of threats to the reliability. Other researchers may possibly come up with different classification schemes, finer or more course grained. However, the consistency of the classification is ensured by having the classifications conducted by the first author and validated by the second.

Internal validity is concerned with the analysis of the data. Since the analysis only uses descriptive statistics, the threats are minimal. Finally, external validity is about generalization from this study. Since we do not draw any conclusions about mapping studies in general, but only on this specific one, the external validity threats are not applicable.

3 Challenges in testing a software product line

Software product line engineering is a development paradigm based on common software platforms, which are customized in order to form specific products [59]. A *software platform* is a set of *generic components* that form a common structure, from which a set of derivative products can be developed [46]. The process of developing the platform is named *domain engineering*, and the process of deriving specific products from the platform is named *application engineering* [59]. We refer to domain testing and application testing, accordingly. The variable characteristics of the platform, are referred to as *variability*; the specific representations of the variability in software artifacts are called *variation points*, while the representation of a particular instance of a variable characteristic is called a *variant* [59].

A number of challenges regarding testing of software product lines have been identified and discussed in the literature, which are identified in this mapping study (RQ1). They can be summarized in three main challenges concerning i) how to handle the large number of tests, ii) how to balance effort for reusable components and concrete products, and iii) how to handle variability.

3.1 Large number of tests

A major challenge with testing a software product line regards the large number of required tests. In order to fully test a product line, all possible uses of each generic component, and preferably even all possible product variants, need to be tested. The fact that the number of possible product variants grows exponentially with the number of variation points, makes such thorough testing infeasible. Since the number of products actually developed also increases, there is an increased need for system tests as well.

The main issue here is how to reduce redundant testing and to minimize the testing effort through reuse of test artefacts. The close relationship between the developed products and the fact that they are derived from the same specifications indicates an option to reduce the number of tests, due to redundancy. A well defined product line also includes a possibility to define and reuse test artefacts.

3.2 Reusable components and concrete products

The second major challenge, which of course is closely related to the previous, is how to balance effort spent on reusable components and product variants. Which

components should be tested in domain (platform) engineering, and which should be tested in application (product) engineering? [59] A high level of quality is required for the reusable components but still it is not obvious how much the testing of reusable components may help reducing testing obligations for each product. There is also a question of how to test generic components, in which order and in how many possible variants. The planning of the testing activities is also further complicated by the fact that software process is split and testing may be distributed across different parts of the organizations.

3.3 Variability

Variability is an important concept in software product line engineering, and it introduces a number of new challenges to testing. Variability is expressed as variation points on different levels with different types of interdependencies. This raises a question of how different types of variation points should be tested. A new goal for testing is also introduced in the context of variability: the verification of the absence of incorrect bindings of variation points. We have to be sure that features not supposed to be there are not included in the end product. The binding of variation points is also important. Complete integration and system test are not feasible until the variation points are bound. It is also possible to realize the same functionality in different ways and thus a common function in different products may require different tests.

4 Primary studies

Following the method defined in Section 2.2, we ended up in 64 peer reviewed papers, published in workshops, conferences, journals and in edited books (RQ2). The papers are published between 2001 and 2008, and summarized by publication fora in Table 1.

Table 1: Distribution of publication fora

Publication Fora	Type	#
International Workshop on Software Product Line Testing (SPLiT)	Workshop	23
International Workshop on Software Product-family Engineering (PFE)	Workshop	3
Software Product Lines – Research Issues in Engineering and Management	Book chapter	3
Software Product Line Conference (SPLC)	Conference	2
ACM SIGSOFT Software Engineering Notes	Journal	1
Communications of the ACM	Journal	1
Concurrency: Specification and Programming Workshop	Workshop	1
Conference on Composition-Based Software Systems	Conference	1

continued on next page

Table 1 – continued from previous page

Publication Fora	Type	#
Conference on Quality Engineering in Software Technology (CONQUEST)	Industry Conference	1
Development of Component-based Information Systems	Book chapter	1
European Conference on Information Systems, Information Systems in a Rapidly Changing Economy, (ECIS)	Conference	1
European Workshop on Model Driven Architecture with Emphasis on Industrial Application	Workshop	1
Fujaba days	Workshop	1
Fundamental Approaches to Software Engineering (FASE)	Conference	1
Hauptkonferenz Net.ObjectDays	Industry Conference	1
International Computer Software and Applications Conference	Conference	1
International Conference on Advanced Information Systems (CAiSE)	Conference	1
International Conference on Automated Software Engineering (ASE)	Conference	1
International Conference on Computer and Information Technology (ICIT)	Conference	1
International Conference on Engineering of Complex Computer Systems (ICECCS)	Conference	1
International Conference on Software Engineering and Formal Methods (SEFM)	Conference	1
International Conference on Software Reuse (ICSR)	Conference	1
International Symposium on Computer Science and Computational Technology (ISCST)	Conference	1
International Symposium on Empirical Software Engineering (ISESE)	Conference	1
International Symposium on Software Reliability Engineering (ISSRE)	Conference	1
International Symposium on Software Testing and Analysis (ISSTA)	Conference	1
International Workshop on Requirements Engineering for Product Lines (REPL)	Workshop	1
International Workshop on Software Product Family Engineering (PFE)	Workshop	1
International Workshop on Product Line Engineering The Early Steps: Planning, Modeling, and Managing (PLEES)	Workshop	1
International Workshop on Software Product Lines	Workshop	1
International Workshop on Test and Analysis of Component Based Systems (TaCOS)	Workshop	1
Journal of Software	Journal	1
Nordic Workshop on Programming and Software Development Tools and Techniques (NWPER)	Workshop	1
The European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)	Conference	1
The Role of Software Architecture for Testing and Analysis (ROSATEA)	Workshop	1
Workshop on Advances in Model Based Testing (A-MOST)	Workshop	1
Workshop on Model-based Testing in Practice	Workshop	1
Total		64

Table 2: Distribution over research focus

Research Focus	2001	2002	2003	2004	2005	2006	2007	2008	Total
Test Organization and Process	1	1	1	2	1	1	1	2	10
Test Management			2	3	1	3	2	4	15
Testability				1		1			2
System and Acceptance Testing		1	4	4	3	7	2	5	26
Integration Testing				1		1	2		4
Unit Testing			2				1		3
Automation				4	1				5
Total	1	2	9	15	6	13	8	11	65

Table 3: Distribution over publication types

Type of Publication	2001	2002	2003	2004	2005	2006	2007	2008	Total	
Book Chapter						4			4	6%
Conference Paper			4	1	2	3	4	5	19	30%
Journal Paper				1		1		1	3	5%
Workshop Paper	1	2	5	13	4	4	4	5	38	59%
Total	1	2	9	15	6	12	8	11	64	100%

In Table 2 and Table 3, the distribution over time is reported for the 64 primary studies. Note that one paper spans two research foci according to our classification scheme. Hence the total number of classification items in Table 2 is 65.

5 Classification Schemes

Publications are classified into categories in three different dimensions: *research focus*, *type of contribution* and *research type*. This structure is presented by Petersen et al. [58]. However the different categories are adapted to this particular study. Establishing the scheme and mapping publications was done iteratively as new primary studies were added. When the scheme was finally set, all classifications were reviewed again.

Six categories of research focus (RQ3) were identified through the keyword method described by Petersen et al. [58]: *i) test organization and process*, *ii) test management*, *iii) testability*, *iv) system and acceptance testing (ST and AT)*, *v) integration testing (IT)*, *vi) unit testing (UT)*, and *vii) automation*. *Test organization and process* includes publications with a focus on the testing framework, seeking answers to how the testing activities and test assets should be mapped to the overall product line development and also how product line testing should be organized

overall. Papers on product line testing in general are also mapped into this category. *Test management* includes test planning and assessment, fault prediction, selection of test strategies, estimates of the extent of testing and test coverage. Papers on how to distribute resources (between domain engineering process and application engineering process, between different test activities, and between different products) are included as well. *Testability* includes papers with a focus on other aspects of product line engineering rather than the testing, but still with the goal of improved testing. The test levels used in the classification are *system and acceptance testing*, *integration testing*, and *unit testing*. Paper topics cover both design of new test cases and selection of already existing test cases. Test cases could be designed from requirements or from generic test assets. Some papers focus on the *automation* of testing.

Contribution type is classified into five categories: *Tool*, *Method*, *Model*, *Metric*, and *Open items*. *Tools* refer to any kind of tool support for SPL testing, mostly in the form of research prototypes. *Methods* include descriptions of how to perform SPL testing, both as general concepts and more specific and detailed working procedures. *Models* are representations of information to be used in SPL testing. *Metrics* focus on what to measure to characterize certain properties of SPL testing. Finally, *Open items* are identified issues that need to be addressed.

The classification of research types (RQ4) is based on a scheme proposed by Wieringa et al. [78]. Research is classified into six categories: *i) validation research*, *ii) evaluation research*, *iii) solution proposals*, *iv) conceptual proposals*, *v) opinion papers*, and *vi) experience papers*. *Validation research* focuses on investigating a proposed solution which has not yet been implemented in practice. Investigations are carried out systematically and include: experiments, simulation, prototyping, mathematical systematically analysis, mathematical proof of properties etc. *Evaluation research* evaluates a problem or an implemented solution in practice and includes case studies, field studies, field experiments etc. A *Solution proposal* is a novel or significant extension to an existing technique. Its benefits are exemplified and/or argued for. A *Conceptual proposal* sketches a new way of looking at things, but without the preciseness of a solution proposal. *Opinion papers* report on the authors' opinions on what is good or bad. *Experience papers* report on personal experiences from one or more real life projects. Lessons learned are included but there is no systematic reporting of research methodology.

6 Mapping

Figure 2 shows a map over existing research foci related to software product line testing, distributed over type of research and type of contribution. The number of publications on each side differs, since some publications provide multiple contributions e.g. both a model and a method. Most research effort is spent on system testing with contributions such as proposed methods for test case design, sketched

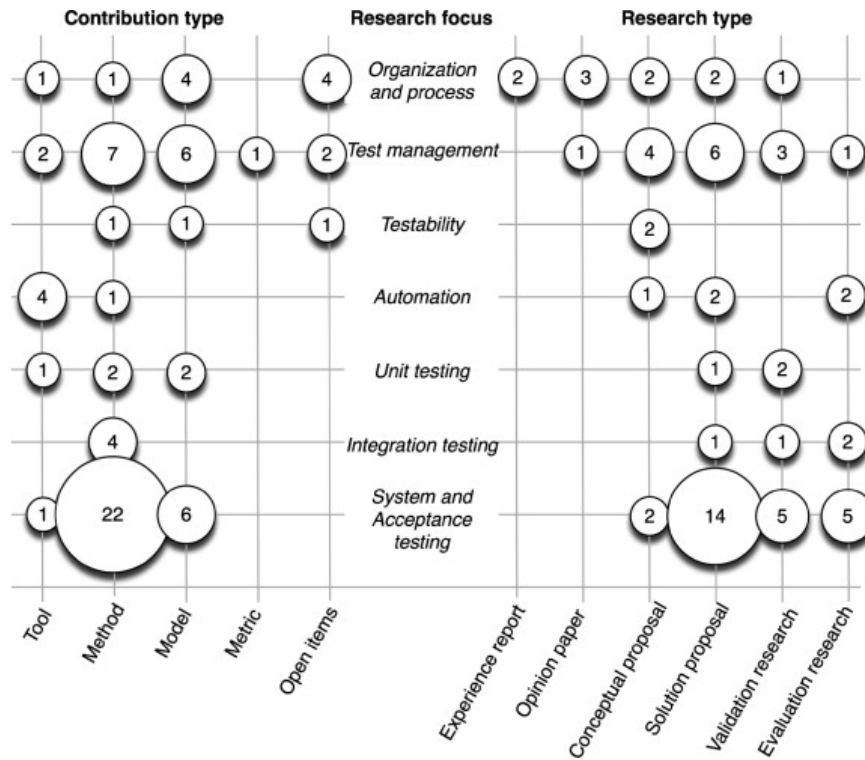


Figure 2: Map of research focus on software product line testing. Research focus on the Y axis; contribution type on the left side of the X axis, and research type on the right side of the X axis

out in detail but not yet evaluated, i.e. solution proposals. An overview of research presented by focus is given in sections 6.1.1 - 6.1.7.

6.1 Research focus

Figure 3 shows the distribution of research foci. A paper is assigned to several foci if it has a clear contribution to more than one area. Each of the focus areas is discussed below.

Test Organization and Process

Table 4 lists all papers on test organisation and process. McGregor points out the need for a well designed test process, and discusses the complex relationships between platforms, products and different versions of both platforms and products

Table 4: Papers on test organization and process

Author	Title	Paper type	Contribution type
Shaulis [68]	Salion's Confident Approach to Testing Software Product Lines	Experience report	Tool
Knauber and Hetrick [32]	Product Line Testing and Product Line Development - variations on a Common Theme	Solution proposal	Method
McGregor [41]	Structuring Test Assets in a Product Line Effort	Conceptual proposal	Model
Weingärtner [76]	Product family engineering and testing in the medical domain-validation aspects	Opinion	Model
Ganesan et al. [18]	Comparing Costs and Benefits of Different Test Strategies for a Software product Line: A study from Testo AG	Validation research	Model
Jin-hua et al. [25]	The W-Model for Testing Software Product Lines	Solution Proposal	Model
Kolb, Muthig [35]	Challenges in Testing Software Product Lines	Opinion paper	Open Items
Tevanlinna, Taina, Kauppinen [72]	Product Family Testing - a Survey	Opinion paper	Open Items
Kolb, Muthig [37]	Techniques and Strategies for Testing component-Based Software and Product Lines	Experience Report	Open Items
Ghanam et al. [20]	A Test-Driven Approach to Establishing & Managing Agile Product Lines	Conceptual proposal	Open Items

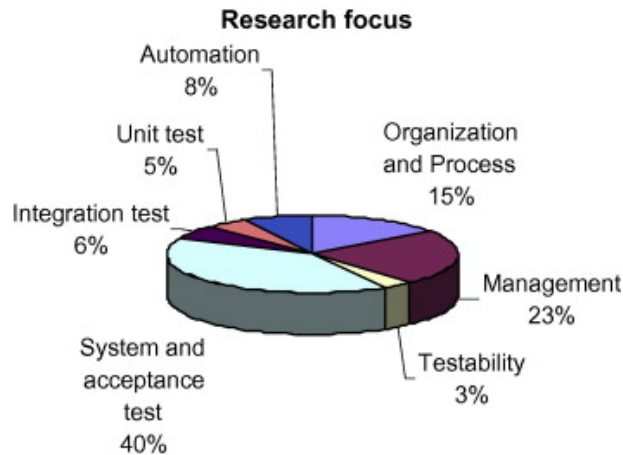


Figure 3: Distribution of research foci

in his technical report [42]. He argues there and elsewhere [41] for a structure of test assets and documentation in alignment with the structure of the constructed products. This is further concretized by Knauber and Hetrick [32]. Kolb and Muthig [35,37] discuss the importance and complexity of testing a software product line and component-based systems. They pinpoint the need for guidelines and comprehensive and efficient techniques for systematically testing product lines. They also promote the idea of creating generic test cases. Tevalinna et al. address the problem of dividing product line testing into two distinct instantiations of the v-model; testing is product oriented and no efficient techniques for domain testing exist [72]. Two problems are pointed out: First, complete integration and system testing in domain engineering is not feasible, and second, it is hard to decide how much we can depend on domain testing in the application testing. They also discuss four different strategies to model product line testing: testing product by product, incremental testing of product lines, reusable asset instantiation and division of responsibilities [72]. Weingärtner discusses the application of product family engineering in an environment where development was previously done according to the V-model [76]. Jin-hua et al. proposes a new test model for software product line testing, the W-model [25]. Ganesan et al. [18] compare cost benefits of a product focused test strategy contra an infrastructure focused test strategy and introduces a cost model to be able to quantify the influences on test costs from a given product variant. Ghanam et al. [20] discuss testing in the context of agile PL and highlights challenges in applying test driven development (TDD) in SPL. Shalius reports on positive experiences of agile testing in the context of XP and RUP [68]

Test Management

The research on test management contains several proposals and a few evaluated research statements, see Table 5. Tevanlinna proposes a tool, called RITA (fRamework Integration and Testing Application) to support testing of product lines [73]. Kolb presents a conceptual proposal that sets focus on test planning and test case design, based on risks [34]. Mc Gregor and Im make a remark that product lines vary both in space and in time, and outline a conceptual proposal to address this fact [44]. Oster et al. proposes a story driven approach to select which features to be tested in different product instances [57].

McGregor discusses, in his technical report, the possibility of product line organizations to retrieve a high level of structural coverage by aggregating the test executions of each product variant in the product line [42]. Schneidemann optimized product line testing by minimizing the number of configurations needed to verify the variation of the platform [67]. Gustafsson worked on algorithms to ensure that all features of a product line are covered in at least one product instance [22]. Cohen et al. [8] define a family of cumulative coverage criteria based on a relational model capturing variability in the feasible product variants, e.g. the orthogonal variability model. Kauppinen et al. propose special coverage criteria for product line frameworks [29].

In order to reduce the test effort, McGregor proposes a combinatorial test design where pairwise combinations of variants are systematically selected to be tested instead of all possible combinations [42]. Muccini and van der Hoek [48] propose a variant of this approach for integration testing, “core first then big bang”, and emphasize the need for a combination of heuristic approaches to combine in order to effectively perform integration testing. Cohen et al. [8] propose application of interaction testing and connect this to the combinatorial coverage criteria.

Al Dallal and Sorenson present a model that focuses on framework testing in application engineering [10]. They identify uncovered framework use cases and select product test cases to cover those. The model is empirically evaluated on software, some 100 LOC in size.

Zeng et al. identify factors that influence SPL testing effort, and propose cost models accordingly [80]. Dowie et al. evaluate different approaches to SPL testing, based on a theoretical evaluation framework [13]. They conclude that the customer’s perspective is missing in SPL testing, and must be included to make the approach successful.

Jaring et al. propose a process model, called VTIM (Variability and Testability Interaction Model) to support management of trade-offs on the binding point for a product line instance [24]. They illustrate the model on a large-scale industrial system. Denger and Kolb report on a formal experiment, investigating inspection and testing as means for defect detection in product line components [11]. Inspections were shown to be more effective and efficient for that purpose. Mc Gregor [43] discusses the need for more knowledge on faults likely to appear in a product line

Table 5: Papers on Test Management

Author	Title	Paper type	Contribution type
Tevanlinna [73]	Product family testing with RITA	Solution Proposal	Tool
Kolb [34]	A Risk-Driven Approach for Efficiently Testing Software Product Lines	Solution Proposal	Method
Scheidemann [67]	Optimizing the selection of representative Configurations in Verification of Evolving Product Lines of Distributed Embedded Systems	Solution Proposal	Method
Gustafsson [22]	An Approach for Selecting Software Product Line Instances for Testing	Validation Research	Method
McGregor and Im [44]	The Implications of Variation for Testing in a Software Product Line	Conceptual Proposal	Method
Oster et al. [57]	Towards Software Product Line Testing using Story Driven Modeling	Conceptual Proposal	Method
Cohen et al. [8]	Coverage and Adequacy in Software Product Line Testing	Solution Proposal	Model, Method
Al Dallal and Sorenson [10]	Testing software assets of framework-based product families during application engineering stage	Validation Research	Model, method, tool
Zeng et al. [80]	Analysis of Testing Effort by Using Core Assets in Software Product Line Testing	Solution Proposal	Model
Dowie et al. [13]	Quality Assurance of Integrated Business Software: An Approach to Testing Software Product Lines	Solution Proposal	Model
Jaring et al. [24]	Modeling Variability and Testability Interaction in Software Product Line Engineering	Evaluation Research	Model
McGregor [43]	Toward a Fault Model for Software Product Lines	Conceptual Proposal	Model
Kauppinen et al. [29]	Hook and Template Coverage Criteria for Testing Framework-based Software Product Families	Conceptual Proposal	Metric
Denger and Kolb [11]	Testing and Inspecting Reusable Product Line Components: First Empirical Results	Validation Research	Open Items
Muccini and van der Hoek [48]	Towards Testing Product Line Architectures	Opinion Paper	Open Items

Table 6: Papers on Testability

Author	Title	Paper type	Contribution type
Kolb and Muthig [36]	Making Testing Product Lines More Efficient by Improving the Testability of Product Line Architectures	Conceptual Proposal	Model, Method
Trew [74]	What Design Policies Must Testers Demand from Product Line Architects?	Conceptual Proposal	Open Items

instance, and outlines a fault model. Fault models may be used as a basis for test case design and as help in estimating required test effort to detect a certain class of faults.

Testability

McGregor discusses testability of software product lines in his technical report [42]. This refers to technical characteristics of the software product that helps testing. We identified two papers on testability, see Table 6. Trew [74] identifies classes of faults that cannot be detected by testing and claim the need for design policies to ensure testability of an SPL. Kolb and Muthig [36] discuss the relationships between testability and SPL architecture and propose an approach to improve and evaluate testability.

System and Acceptance Testing

Table 7 lists paper on system and acceptance testing. Most research effort is spent on system and acceptance testing, 40 %. The most frequent goal is automatic generation of test cases from requirements. Requirements may be model based, mostly on use cases [62], formal specifications [47] or written in natural language [3].

Hartman et al. present an approach based on existing UML based tools and methods [23]. Bertolino and Gnesi introduce PLUTO, product line use case test optimization [4, 6], which is further elaborated by Bertolini et al. [4]. Kamsties et al. propose test case derivation for domain engineering from use cases, preserving the variability in the test cases [27].

Nebut et al. propose an algorithm to automatically generate product-specific test cases from product family requirements, expressed in UML [51, 52], more comprehensively presented in [50]. They evaluate their approach on a small case study. Reuys et al. defined the ScenTED approach to generate test cases from UML models [64], which is further presented by Pohl and Metzger [60]. Olimpiew and Gomaa defined another approach using diagrams, stereotypes and tagged values from UML notations [53, 56] which was illustrated in a student project [55]. Dueñas et al. propose another approach, based on the UML testing profile [14]

Table 7: Papers on System and Acceptance Testing

Author	Title	Paper type	Contribution type
Hartmann et al. [23]	UML-based approach for validating product lines	Solution Proposal	Tool
Bertolino and Gnesi [5]	Use Case-based Testing of Product Lines	Solution Proposal	Method
Bertolino and Gnesi [6]	PLUTO: A test Methodology for product Families	Validation Research	Method
Kamsties et al. [27]	Testing Variabilities in Use case Models	Solution Proposal	Method
Nebut et al. [51]	Automated Requirements-based Generation of Test Cases for Product Families	Validation Research	Method
Stephenson et al. [71]	Test Data Generation for Product Lines - A Mutation Testing Approach	Solution Proposal	Method
Geppert et al. [19]	Towards Generating Acceptance Tests for Product Lines	Validation Research	Method
Olimpiew and Gomaa [53]	Model-based Testing for Applications Derived from Software Product Lines	Solution Proposal	Method
Reuys et al. [64]	Model-Based System Testing of Software Product Families	Evaluation Research	Method
Mishra [47]	Specification Based Software Product Line Testing: A case study	Solution Proposal	Method
Olimpiew and Gomaa [55]	Customizable Requirements-based Test Models for Software Product Lines	Evaluation Research	Method
Pohl and Metzger [60]	Software Product Line Testing	Conceptual Proposal	Method
Reis et al. [62]	A Reuse Technique for Performance Testing of Software Product Lines	Evaluation Research	Method
Reuys et al. [66]	The ScenTED Method for Testing Software Product Lines	Evaluation Research	Method
Li et al. [39]	Reuse Execution Traces to Reduce Testing of Product Lines	Evaluation Research	Method
Bashardoust-Tajali and Corriveau [3]	On extracting Tests from a Testable Model in the Context of Domain Engineering	Solution Proposal	Method
Kahsai et al. [26]	Specification-based Testing for Software Product Lines	Solution Proposal	Method
Olimpiew and Gomaa [56]	Model-Based Test Design for Software Product Lines	Solution Proposal	Method
Uzuncaova et al. [75]	Testing Software Product Lines Using Incremental Test Generation	Validation Research	Method
Weißleder et al. [77]	Reusing State Machines for Automatic Test Generation in Product Lines	Solution Proposal	Method
Dueñas et al. [14]	Model driven testing in product family context	Solution Proposal	Model
Nebut et al. [50]	System Testing of Product Lines: From Requirements to Test Cases	Validation Research	Model
Olimpiew and Gomaa [54]	Reusable System Tests for Applications Derived from Software Product Lines	Conceptual Proposal	Model
Kang et al. [28]	Towards a Formal Framework for Product line Test Development	Solution Proposal	Model, Method
Nebut et al. [52]	Reusable Test Requirements for UML-Model Product Lines	Solution Proposal	Model, Method
Bertolino et al. [4]	Product Line Use Cases: Scenario-Based Specification and Testing of Requirements	Solution Proposal	Model, Method

Table 8: Papers on Integration Testing

Author	Title	Paper type	Contribution type
Reuys et al. [66]	The ScenTED Method for Testing Software Product Lines	Evaluation Research	Method
Kishi and Noda [30]	Design Testing for Product Line Development based on Test Scenarios	Solution Proposal	Method
Li et al. [40]	Automatic Integration Test Generation from Unit Tests of eXVantage Product Family	Evaluation Research	Method
Reis et al. [63]	Integration testing in software product line engineering: A model-Based Technique	Validation Research	Method

and Kang et al. yet another process, based on UML use cases and a variability model [28]. Weißleder et al. specifically reuse state machines and generate sets suites, using OCL expressions [77].

Mishra [47] and Kahsai et al. [26] present test case generation models, based on process algebra formal specifications. Uzuncanova et al. introduce an incremental approach to test generation, using Alloy [75]. Bashardoust-Tajali and Corriveau extract tests for product testing, based on a domain model, expressed as generative contracts [3].

Stephensen et al. propose a test strategy to reduce the search space for test data, although without providing any reviewable details [71]. Geppert et al. present a decision model for acceptance testing, based on decision trees [19]. The approach was evaluated on a part of an industrial SPL. Li et al. utilize the information in execution traces to reduce test execution of each product of the SPL [39].

Integration Testing

Table 8 lists papers on integration testing. The ScenTED method is proposed also for integration testing in addition to system and acceptance testing, and hence mentioned here [66]. Reis et al. specifically validated its use for integration testing in an experimental evaluation [63]. Kishi and Noda propose an integration testing technique based on test scenarios, utilizing model checking techniques [30]. Li et al. generate integration test from unit tests, illustrated in an industrial case study [40].

Unit Testing

Table 9 lists papers on unit testing. Different approaches to create test cases based on requirements including variabilities, are proposed with a focus on how to cover possible scenarios. In ScenTED, [65], UML-activity diagrams are used to represent all possible scenarios. Nebut et al. [49] use parameterized use cases as

Table 9: Papers on Unit Testing

Author	Title	Paper type	Contribution type
Feng et al. [16]	A product line based aspect-oriented generative unit testing approach to building quality components	Validation Research	Method
Reuys et al. [65]	Derivation of Domain Test Scenarios from Activity Diagrams	Solution Proposal	Model
Nebut et al. [49]	A Requirement-Based Approach to test Product Families	Validation Research	Model, Method, Tool

Table 10: Papers on Test Automation

Author	Title	Paper type	Contribution type
Knauber and Schneider [33]	Tracing Variability from Implementation to Test Using Aspect-Oriented Programming	Conceptual Proposal	Tool
Williams [79]	Test Case Management of Controls Product Line Points of Variability	Solution Proposal	Tool
Condron [9]	A Domain Approach to Test Automation of Product Lines	Solution Proposal	Tool
Ganesan et al. [17]	Towards Testing Response time of Instances of a web-based Product Line	Evaluation Research	Tool
McGregor et al. [45]	Testing Variability in a Software Product Line	Evaluation Research	Method

contracts on which testing coverage criteria may be applied. Feng et al. use an aspect-oriented approach to generate unit tests [16].

Test Automation

Table 10 lists papers on test automation. McGregor et al. [45] propose and evaluate an approach to design test automation software which is based on correspondence between variability in product software and in test software. Condron [9] proposes a domain approach to automate PL testing, combining test automation frameworks from various locations in the entire product line where test is needed. Knauber and Schneider [33] explore how to combine aspect oriented programming and unit testing and thus reach traceability between implementation of variability and its test. Ganesan et al. [17] focus on performance testing, reporting on a realization of an environment for testing response time and load of an SPL. Williams presents an approach to integrating test automation in an existing development environment for control systems [79].

6.2 Research Type

Figure 4, shows the distribution of research types in the area of software product line testing. The most frequent research type is solution proposals 41 %. Adding solution, conceptual proposals and opinion papers sum up to 64 % of the papers. 14 % of the papers report on evaluation of the proposals and 3 % are experience reports. 19 % present other types of validation, primarily off-line approaches.



Figure 4: Distribution of Research Type

7 Discussion

The surveyed research indicates software product line testing being a rather immature area. The seminal paper is presented in 2001 [42], and most papers are published in workshops and conferences; only one has reached the maturity of a journal publication.

Software product line testing seems to be a “discussion” topic. There is a well established understanding about challenges, as summarized in Section 6. However, when looking for solutions to these challenges, we mostly find proposals. The mapping shows that 64 % of the papers found include proposals, which contain ideas for solutions of the identified challenges, but only 17 % of the research report actual use and evaluation of proposals.

This is not unique for the SPL testing. Ramesh et al. reviewed publications in 13 computer science journals, and found less than 3 % being case studies, field studies or experiments [61]. Close to 90 % were of research type “conceptual analysis”, which is close to our “proposals” categories. In software engineering, the case is somewhat better. Glass et al. reported 2002 that “conceptual analysis”

also dominates in software engineering (54 %), while case study, field study and experiment sum up to less than 10 % [21].

Product line testing is a large scale effort and evaluations are costly [72], which is one of the explanations behind the limited share of empirical studies. However, extensive experience in PL engineering exist within companies (Philips, Nokia, Siemens etc. [59]) but no studies on testing can be found [72]. The distribution across the research foci, with its major share on system testing is natural. This is where product line testing may gain a lot from utilizing the fact that it is a software product line. Testability issues, especially related to the product line architecture have an underdeveloped potential to be researched. Approaches that help isolate effects of variability to limited areas of the software would help improve the efficiency of product line testing. Test management issues have a reasonable proportion of the studies, although issues of balancing e.g. domain vs. product testing are not treated. Some sketched out proposals and many high-level opinions on how this should be done are reported on but none of them has been evaluated empirically.

Almost all of the proposed strategies for product line testing are idealistic in the sense that they put specific requirements on other parts of the development process than the testing. Hence, it is hard to find “useful approaches”, since they require major changes to the whole software engineering process, e.g. formal models for requirements and variability. In a majority of the publications the handling of variability is in focus. Different approaches for test case derivation are based on specific ways of documenting and handling variation points. This is natural since variability is the core concept in product line development. However from the perspective of system testing the main challenge is how to deal with the large number of required tests of a range of product variants which are more or less similar. How variability is handled may not always be possible to affect or even visible at that stage. There is a need for strategies for test case design and selection, which are feasible for incremental introduction and applicable in a testing context regardless of the maturity of the product line organization.

The contribution type is mostly of “method” type. Product line engineering in general, and testing in particular, need new methodological approaches. However, methods need to be supported by underlying models for their theoretical foundation, tools for their practical use and metrics for their management and evaluation.

8 Conclusions

We launched a systematic mapping study to get an overview of existing research on software product line testing. We identified 64 papers published between 2001 and 2008.

The picture of research needs and challenges is quite clear and unanimous, enabling a focused research endeavor. In response to RQ 1, the main challenges

are i) the large number of tests, ii) balance between effort for reusable components and concrete products, and iii) handling variability. Still, there is a need to address different focus: process and organization, management, testability, test case design as well as test automation. To respond to RQ2, we conclude that the research is mostly published in workshops (59 %) and conferences (30 %), with only four book chapters and three journal publications issued so far. The research topics identified are (RQ3) i) test organization and process, ii) test management, iii) testability, iv) system and acceptance testing, v) integration testing, vi) unit testing, and vii) automation, with high-level test case derivation as the most frequent topic followed by test management. Research methods (RQ4) are mostly of proposal type (64 %) with empirical evaluations and experience as a minor group (17 %).

With a clear picture of needs and challenges, we encourage the research community to launch empirical studies that use and evaluate the proposals, in order to give a solid foundation for software product line testing in industry. Further, trade-off management issues seem to be in need of deeper understanding and evaluation.

References

- [1] Wasif Afzal, Richard Torkar, and Robert Feldt. A systematic review of search-based testing for non-functional system properties. *Information and Software Technology*, 51(6):957–976, June 2009.
- [2] John Bailey, David Budgen, Mark Turner, Barbara A. Kitchenham, Pearl Brereton, and Stephen Linkman. Evidence relating to object-oriented software design: A survey. In *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, pages 482–484, September 2007.
- [3] Soheila Bashardoust-Tajali and Jean-Pierre Corriveau. On extracting tests from a testable model in the context of domain engineering. In *Proceedings of the IEEE 13th International Conference on Engineering of Complex Computer Systems (ICECCS 2008)*, pages 98–107, April 2008.
- [4] Antonia Bertolino, Alessandro Fantechi, Stefania Gnesi, and Giuseppe Lami. Product line use cases: Scenario-based specification and testing of requirements. In *Software Product Lines - Research Issues in Engineering and Management*, pages 425–445. Springer-Verlag, 2006.
- [5] Antonia Bertolino and Stefania Gnesi. Use case-based testing of product lines. In *Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering (ESEC/FSE-11)*, pages 355–358, 2003.
- [6] Antonia Bertolino and Stefania Gnesi. PLUTO: a test methodology for product families. In *Software Product Family Engineering Software Product Family Engineering*, volume 3014 of *Lecture Notes in Computer Science*, pages 181–197. Springer-Verlag, 2004.
- [7] Jan Bosch. *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. Addison-Wesley, 2000.
- [8] Myra B. Cohen, Matthew B. Dwyer, and Jiangfan Shi. Coverage and adequacy in software product line testing. In *Proceedings of the Workshop on Role of software architecture for testing and analysis (ROSATEA '06)*, pages 53–63, 2006.
- [9] Chris Condron. A domain approach to test automation of product lines. In *Proceedings International Workshop on Software Product Line Testing (SPLiT)*, Technical Report: ALR-2004-031, pages 27–35. Avaya Labs, 2004.
- [10] Jehad Al Dallal and Paul Sorenson. Testing software assets of framework-based product families during application engineering stage. *Journal of Software*, 3(5):11–25, 2008.

- [11] Christian Denger and Ronny Kolb. Testing and inspecting reusable product line components: first empirical results. In *Proceedings of the ACM/IEEE international symposium on Empirical software engineering (ISESE '06)*, pages 184–193, 2006.
- [12] Oscar Dieste, Anna Grimán, and Natalia Juristo. Developing search strategies for detecting relevant experiments. *Empirical Software Engineering*, 14(5):513–539, October 2009.
- [13] Ulrike Dowie, Nicole Gellner, Sven Hanssen, Andreas Helferich, and Georg Herzworm. Quality assurance of integrated business software: An approach to testing software product lines. In *Proceedings of the 13th European Conference on Information Systems, Information Systems in a Rapidly Changing Economy*, ECIS, 2005.
- [14] Juan C. Dueñas, Julio Mellado, Rodrigo Cerón, José L. Arciniegas, José L. Ruiz, and Rafael Capilla. Model driven testing in product family context. In *University of Twente*, 2004.
- [15] Emelie Engström, Per Runeson, and Mats Skoglund. A systematic review on regression test selection techniques. *Information and Software Technology*, 52(1):14–30, January 2010.
- [16] Yankui Feng, Xiaodong Liu, and Jon Kerridge. A product line based aspect-oriented generative unit testing approach to building quality components. In *Proceedings of the 31st Annual International Computer Software and Applications Conference*, pages 403–408, July 2007.
- [17] Dharmalingam Ganesan. Towards testing response time of instances of a web-based product line. In *Proceedings of the International Workshop on Software Product Line Testing (SPLiT 2005)*, pages 23–34, 2005.
- [18] Dharmalingam Ganesan, Jens Knodel, Ronny Kolb, Uwe Haury, and Gerald Meier. Comparing costs and benefits of different test strategies for a software product line: A study from testo AG. In *Proceedings of the International Software Product Line Conference*, pages 74–83, 2007.
- [19] Birgit Geppert, Jenny Li, Frank Rößler, and David M. Weiss. Towards generating acceptance tests for product lines. In *Software Reuse: Methods, Techniques, and Tools*, volume 3107 of *Lecture Notes in Computer Science*, pages 35–48. Springer Berlin Heidelberg, 2004.
- [20] Yaser Ghanam, Shelly Park, and Frank Maurer. A test-driven approach to establishing & managing agile product lines. In *Proceedings of the Fifth International Workshop on Software Product Line Testing (SPLiT 2008)*, page 46, 2008.

-
- [21] Robert L. Glass, Iris Vessey, and Venkataraman Ramesh. Research in software engineering: an analysis of the literature. *Information and Software Technology*, 44(8):491–506, 2002.
- [22] Thomas Gustafsson. An approach for selecting software product line instances for testing. pages 81–86, 2007.
- [23] Jean Hartmann, Marlon Vieira, and Axel Ruder. A UML-based approach for validating product lines. In *Proceedings of the International Workshop on Software Product Line Testing (SPLiT 2004)*, pages 58–65, August 2004.
- [24] Michel Jaring, René L. Krikhaar, and Jan Bosch. Modeling variability and testability interaction in software product line engineering. In *Seventh International Conference on Composition-Based Software Systems (ICCBSS)*, pages 120–129, 2008.
- [25] Li Jin-hua, Li Qiong, and Li Jing. The w-model for testing software product lines. In *Computer Science and Computational Technology, International Symposium on*, volume 1, pages 690–693, Los Alamitos, CA, USA, 2008. IEEE Computer Society.
- [26] Temesghen Kahsai, Markus Roggenbach, and Bernd-Holger Schlingloff. Specification-based testing for software product lines. In *Proceedings of the IEEE Sixth International Conference on Software Engineering and Formal Methods (SEFM'08)*, pages 149–158, 2008.
- [27] Erik Kamsties, Klaus Pohl, Sacha Reis, and Andreas Reuys. Testing variabilities in use case models. In *Proceedings of the 5th International Workshop on Software Product-Family Engineering (PFE-5)*, pages 6–18, November 2003.
- [28] Sungwon Kang, Jihyun Lee, Myungchul Kim, and Woojin Lee. Towards a formal framework for product line test development. In *Proceedings of the Computer and Information Technology*, pages 921–926, 2007.
- [29] Raine Kauppinen, Juha Taina, and Antti Tevanlinna. Hook and template coverage criteria for testing framework-based software product families. In *Proceedings of the International Workshop on Software Product Line Testing (SPLIT)*, pages 7–12, 2004.
- [30] Tomoji Kishi and Natsuko Noda. Design testing for product line development based on test scenarios. In *Proceedings of the International Workshop on Software Product Line Testing (SPLiT 2004)*, pages 19–26, August 2004.
- [31] Barbara Kitchenham. Guidelines for performing systematic literature reviews in software engineering. Technical report, and Department of Computer Science, University of Durham, Version 2.3, 2007.

- [32] Peter Knauber and W. A. Hetrick. Product line testing and product line development - variations on a common theme. In *Proceedings of the International Workshop on Software Product Line Testing (SPLiT 2005)*, 2005.
- [33] Peter Knauber and Johannes Schneider. Tracing variability from implementation to test using aspect-oriented programming. In *Proceedings of the International Workshop on Software Product Line Testing (SPLiT)*, Technical Report: ALR-2004-031, pages 36–44. Avaya Labs, 2004.
- [34] Ronny Kolb. A risk driven approach for efficiently testing software product lines. In *5th GPCE Young, Researches Workshop*, Erfurt, Germany, 2003.
- [35] Ronny Kolb and Dirk Muthig. Challenges in testing software product lines. In *Proceedings of CONQUEST'03*, pages pp. 81–95, Nuremberg, Germany, September 2003.
- [36] Ronny Kolb and Dirk Muthig. Making testing product lines more efficient by improving the testability of product line architectures. In *Proceedings of the ACM ISSTA workshop on Role of software architecture for testing and analysis (ROSATEA '06)*, pages 22–27, 2006.
- [37] Ronny Kolb and Dirk Muthig. Techniques and strategies for testing component-based software and product lines. In *Development of component-based information systems*, number 2 in *Advances in Management Information Systems*, pages 123–139. 2006.
- [38] Beatriz Pérez Lamancha, Macario Polo Usaola, and Mario Piattini Velthius. Software product line testing—a systematic review. pages 23–30, 2009.
- [39] J. Jenny Li, Birgit Geppert, Frank Röbler, and David M. Weiss. Reuse execution traces to reduce testing of product lines. In *Proceedings of the 11th International Conference Software Product Lines. Second Volume (Workshops)*, pages 65–72, 2007.
- [40] J. Jenny Li, David M. Weiss, and J. Hamilton Slye. Automatic integration test generation from unit tests of eXVantage product family. In *Proceedings of the 11th International Conference Software Product Lines. Second Volume (Workshops)*, pages 73–80, 2007.
- [41] John D. McGregor. Structuring test assets in a product line effort. In *Proceedings of the Second International Workshop on Software Product Lines: Economics, Architectures, and Implications*, pages 89–92, May 2001.
- [42] John D. McGregor. Testing a software product line. Technical Report CMU/SEI-2001-TR-022, ESC-TR-2001-022, Software Engineering Institute, Carnegie Mellon University, 2001.

-
- [43] John D. McGregor. Toward a fault model for software product lines. In *Proceedings Fifth International Workshop on Software Product Line Testing*, (SPLiT 2008), 2008.
- [44] John D. McGregor and Kyungsoo Im. The implications of variation for testing in a software product line. In *Proceedings of the International Workshop on Software Product Line Testing (SPLiT 2007)*, 2007.
- [45] John D. McGregor, P. Sodhani, and S. Madhavapeddi. Testing variability in a software product line. In *Proceedings of the international workshop on software product line testing (SPLiT)*, Technical Report: ALR-2004-031, pages 45–50. Avaya Labs, 2004.
- [46] Marc H. Meyer and Alvin P. Lehnerd. *The Power of Product Platforms*. Free Press, reprint edition, 1997.
- [47] Satish Mishra. Specification based software product line testing: A case study. In *Concurrency, Specification and Programming (CS&P 2006)*, 2006.
- [48] Henry Muccini and André Van Der Hoek. Towards testing product line architectures. In *Proceedings of International Workshop on Testing and Analysis of Component Based Systems*, pages 111–121, 2003.
- [49] Clémentine Nebut, Franck Fleurey, Yves Le Traon, and Jean-marc Jézéquel. A requirement-based approach to test product families. *Proceedings of the 5th workshop on product families engineering*, pages 198–210, 2003.
- [50] Clémentine Nebut, Yves Le Traon, and Jean-marc Jézéquel. System testing of product families: from requirements to test cases. *Software Product Lines*, pages 447–478, 2006.
- [51] Clémentine Nebut, Simon Pickin, Yves Le Traon, and Jean-marc Jézéquel. Automated requirements-based generation of test cases for product families. In *Proceedings of the IEEE 18th International Conference on Automated Software Engineering*, pages 263–266, October 2003.
- [52] Clémentine Nebut, Simon Pickin, Yves Le Traon, and Jean-marc Jézéquel. Reusable test requirements for UML-Modeled product lines. In *Proceedings of the Workshop on Requirements Engineering for Product Lines (REPL'02)*, pages 51–56, 2002.
- [53] Erika Mir Olimpiew and Hassan Gomaa. Model-based testing for applications derived from software product lines. In *Proceedings of the 1st international workshop on Advances in model-based testing (A-MOST '05)*, pages 1–7, 2005.
- [54] Erika Mir Olimpiew and Hassan Gomaa. Reusable system tests for applications derived from software product lines. pages 8–15, 2005.

- [55] Erika Mir Olimpiew and Hassan Gomaa. Customizable requirements based test models for software product lines. In *Proceedings of the International Workshop on Software Product Line Testing (SPLiT)*, August 2006.
- [56] Erika Mir Olimpiew and Hassan Gomaa. Model-based test design for software product lines. 2008.
- [57] Sebastian Oster, Andy Schürr, and Ingo Weisemöller. Towards software product line testing using story driven modeling. In *Proceedings of 6th International Fujaba Days*, pages 48–55, 2008.
- [58] Kai Petersen, Robert Feldt, Mujtaba Shahid, and Michael Mattsson. Systematic mapping studies in software engineering. *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, pages 71–80, 2008.
- [59] Klaus Pohl, Günther Böckle, and Frank J. Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag, 1 edition, September 2005.
- [60] Klaus Pohl and Andreas Metzger. Software product line testing. *Communications of the ACM*, 49:78, December 2006.
- [61] V. Ramesh, Robert L. Glass, and Iris Vessey. Research in computer science: an empirical study. *Journal of Systems and Software*, 70(1–2):165–176, February 2004.
- [62] Sacha Reis, Andreas Metzger, and Klaus Pohl. A reuse technique for performance testing of software product lines. In *Proceedings of the international workshop on software product line testing*. Mannheim University of Applied Sciences, 2006.
- [63] Sacha Reis, Andreas Metzger, and Klaus Pohl. Integration testing in software product line engineering: a model-based technique. In *Proceedings of the 10th international conference on Fundamental approaches to software engineering*, pages 321–335, 2007.
- [64] Andreas Reuys, Erik Kamsties, Klaus Pohl, and Sacha Reis. Model-based system testing of software product families. In *Advanced Information Systems Engineering*, volume 3520, pages 519–534. Springer Berlin Heidelberg, 2005.
- [65] Andreas Reuys, Sacha Reis, Erik Kamsties, and Klaus Pohl. Derivation of domain test scenarios from activity diagrams. In *Proceedings of the International Workshop on Product Line Engineering: The Early Steps: Planning, Modeling, and Managing (PLEES'03)*, Erfurt, 2003.

- [66] Andreas Reuys, Sacha Reis, Erik Kamsties, and Klaus Pohl. The ScenTED method for testing software product lines. In *Proceedings of the Software Product Lines - Research Issues in Engineering and Management*, pages 479–520, 2006.
- [67] Kathrin D. Scheidemann. Optimizing the selection of representative configurations in verification of evolving product lines of distributed embedded systems. In *Proceedings of the 10th International on Software Product Line Conference (SPLC '06)*, pages 75–84, 2006.
- [68] Carl L. Shaulis. Salion’s confident approach to testing software product lines. In *Proceedings of International Conference on Product Line Testing (SPLiT 04)*, 2004.
- [69] Mary Shaw. What makes good research in software engineering. *International Journal on Software Tools for Technologyfor Technology Transfer (STTT)*, 4:1–7, 2002.
- [70] Mats Skoglund and Per Runeson. Reference-based search strategies in systematic reviews. In *Proceedings of the 13th international conference on Evaluation and Assessment in Software Engineering (EASE'09)*, pages 31–40, 2009.
- [71] Zoë Stephenson, Yuan Zhan, John Clark, and John Mcdermid. Test data generation for product lines—a mutation testing approach. volume 3154 of *Lecture Notes in Computer Science*. Heidelberg, 2004.
- [72] Antti Tevanlinna. Product family testing with RITA. In *Proceedings of the Eleventh Nordic Workshop on Programming and Software Development Tools and Techniques (NW- PER'2004)*, Turku, Finland, 2004.
- [73] Antti Tevanlinna, Juha Taina, and Raine Kauppinen. Product family testing: a survey. *SIGSOFT Software Engineering Notes*, 29(2):12, March 2004.
- [74] Tim Trew. What design policies must testers demand from product line architects? In *Proceedings of the International Workshop on Software Product Line Testing*, Technical Report: ALR-2004-031, pages 51–57. Avaya Labs, 2004.
- [75] Engin Uzuncaova, Daniel Garcia, Sarfraz Khurshid, and Don Batory. Testing software product lines using incremental test generation. In *Proceedings of the 19th International Symposium on Software Reliability Engineering (IS-SRE 2008)*, pages 249 –258, November 2008.
- [76] Josef Weingärtner. Product family engineering and testing in the medical domain—validation aspects. In *Software Product-Family Engineering*, number 2290 in *Lecture Notes in Computer Science*, pages 383–387. Springer Berlin Heidelberg, January 2002.

- [77] Stephan Weißleder, Dehla Sokenou, and Bernd-Holger Schlingloff. Reusing state machines for automatic test generation in product lines. In *Proceedings 1st Workshop on Model-based Testing in Practice (MoTiP'08)*, pages 19–28, June 2008.
- [78] Roel Wieringa, Neil Maiden, Nancy Mead, and Colette Rolland. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. *Requirements Engineering*, 11(1):102–107, March 2006.
- [79] Jamie J. Williams. Test case management of controls product line points of variability. In *International Workshop on Software Product Line Testing (SPLiT)*, 2004.
- [80] Hui Zeng, Wendy Zhang, and David Rine. Analysis of testing effort by using core assets in software product line testing. In *Proceedings of the International Workshop on Software Product Line Testing*, Technical Report: ALR-2004-031, pages 1–6. Avaya Labs, 2004.

TEST OVERLAY IN AN EMERGING SOFTWARE PRODUCT LINE – AN INDUSTRIAL CASE STUDY

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.

Emelie Engström and Per Runeson

Journal of Information and Software Technology 55(3):582-594, 2013

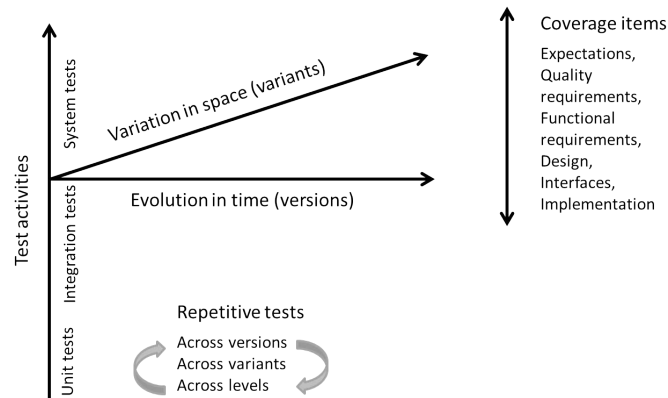


Figure 1: Testing in a product line context is repeated across three dimensions [17]: testing within different test activities (e.g. unit tests, integration tests and system tests), testing across versions (e.g. the continuous software updates), testing of multiple variants (e.g. adaptations to different hardware). Testing at different levels covers the system from different perspectives (e.g. expectations, different types and detail levels of requirements, design or code)

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, since testing is repeated across three dimensions [17]: 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 [6].

To handle the combinatorial explosion of possible tests in software product line development, regression testing approaches are suggested to be applied not only to versions but also to variants [6]. 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 [8, 24]. Most studies are done in the small, and hence questions of scalability and usability are left unanswered [8]. 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 [18].

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 with respect to functionality as well as different purposes of tests. For the assessment of coverage and overlap, 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 [6], testing incrementally evolving software [10], testing of component based systems [23], testing of web based applications [1] and of service oriented architectures [3], as well as from a configuration management perspective [2, 21].

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.* [18], 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 [8] and product line testing [6], conducted a survey on industrial practices [5], and applied regression test selection procedures in-the-small [7, 9].

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 [6]. 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 [20].

Even though regression testing has been researched to a large extent [8, 24], 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 [8]. 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. Applied to a SPL-testing

contexts and according to the 3D model in Figure 1, test results could also be reused across test activities (e.g. between domain testing and application testing as defined by Pohl *et al.* [15]) and variants based on the same assumptions. 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?* – In a SPL context 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.* [25], which is the case for SPL testing. Thus it is relevant to study prerequisites for visualization in this context.

The first research question is mostly *descriptive*, the second is *explanatory*, while the third question comprises an *improving* component [18].

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 [14]. The terminology in the context description is changed to align to generally accepted definitions, if needed.

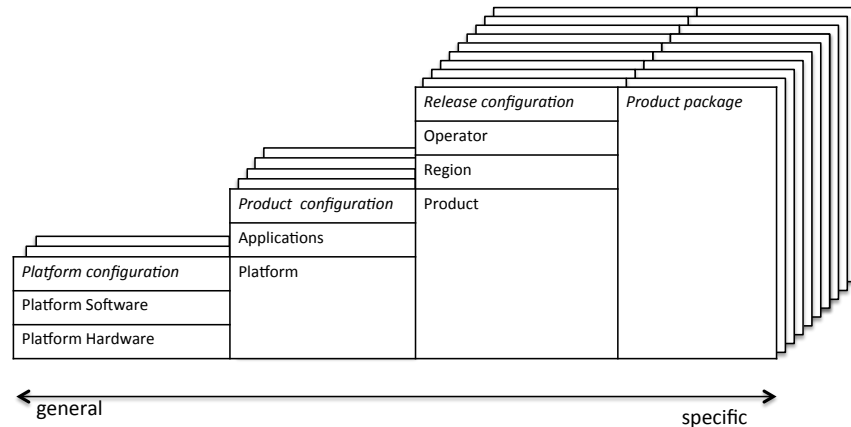


Figure 2: Configuration view of the product line under study.

Products and market

This section describes the different types of variability and commonality in our case. 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 [11], and since the market is very competitive and fast changing, on time delivery is crucial.

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. Table 1 shows the main foci with respect to domain and application testing for the different organizational units. 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 activities* – Testing in each of the core software and application software organizations are conducted in (at least) three main test activities, 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. Domain testing [15] of both platform commonality and product commonality, see Table 1 are mainly conducted within these two units. 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 within all test activities 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.

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.

Table 1: The foci on domain versus application testing [15] within the main test activities of the organizational units. Domain and application testing of platform variants are denoted DomPl and AppPl respectively and corresponding denotations for product variants are DomPr and AppPr respectively.

	Core SW org.	Application SW org.	Product composition org.
Feature testing	DomPl, DomPr	DomPl, DomPr	—
Integration testing	DomPl, DomPr	DomPl, DomPr	—
System testing	AppPl	AppPl	AppPl, DomPr
Acceptance testing	—	—	AppPr

2.4 Case and units of analysis

This study is a single-case study [18] 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 CovIs 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 several superior testing activities (feature tests, integration tests, and system tests) and for each superior test activity, several sub-test activities with differences in scope and focus are carried out, see Figure 3. The sub-units of analysis represents several test activities and organizational units. The activities are selected because of their frequency and costliness in comparison to other test activities within the same superior test activity. 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

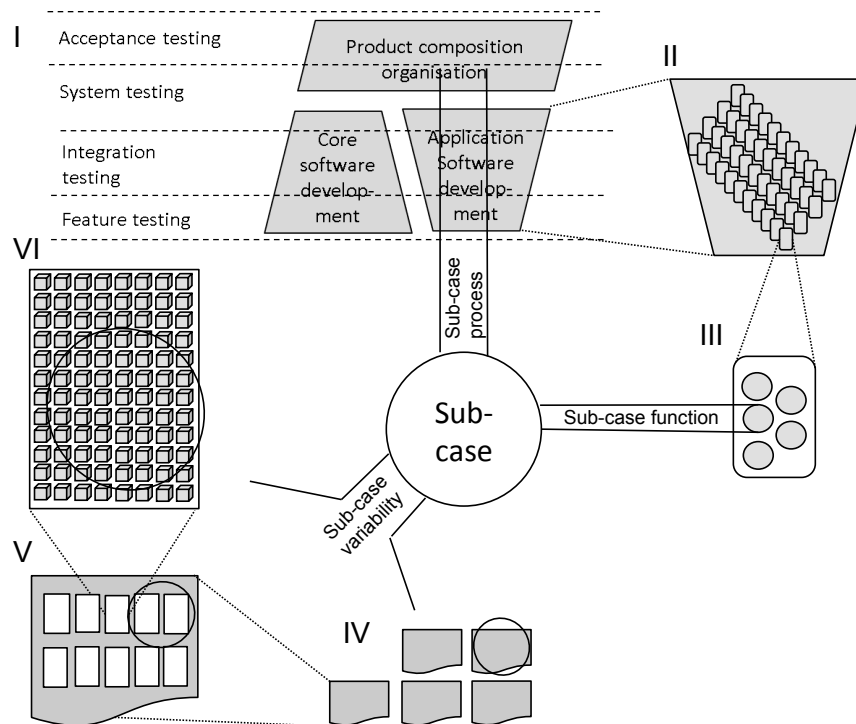


Figure 3: The case covers three organizational units and four superior test activities (I). The scope of the sub-case is delimited to one single function (III) which is one out of five main functions of a feature area (II). Furthermore the sub-case comprises two organizational units and three levels 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).

- Organizational units (core software, application software and product composition)
- Test activities (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 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 [17]:

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 many real-life product line processes. However, some good examples are presented by van der Linden *et al.* [12].

2.6 Propositions

Propositions are predictions about the world that may be deduced logically from theory [19]. 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.* [22] 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 new 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.

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” [4]. An *emerging* software product line is one where the “prescribed way” is not well defined yet.

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

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.

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.

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.

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.

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

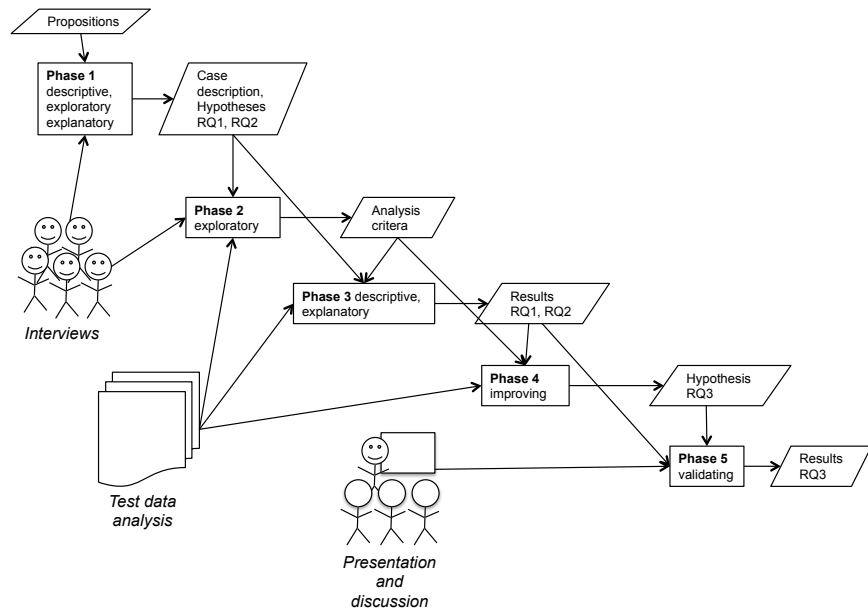


Figure 4: The different phases of the case study

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 2. The interviews had the format of open discussions covering a number of high level topics listed in Table 2. During the interviews notes were taken and some 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

Table 2: 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.: <i>feature testing</i>	Application SW org.: <i>integration testing</i>	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

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 [13].

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 Rune-son *et al.*'s guidelines [18], 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 been 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 are well in line with this, strengthening the reliability of this study.

Internal validity is concerned with causal 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 causal 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 2. 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 product variants 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.

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 3, 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 3.

Table 3: Number of test cases explicitly designed 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	Name	Purpose level 1 (P1)			Purpose level 2 (P2)		Purpose level 3 (P3)	Total
		<i>Interaction</i>	<i>Interoperability</i>	<i>5 quality attributes</i>	<i>10 interacting functions</i>	<i>40 brands</i>	<i>112 models</i>	
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

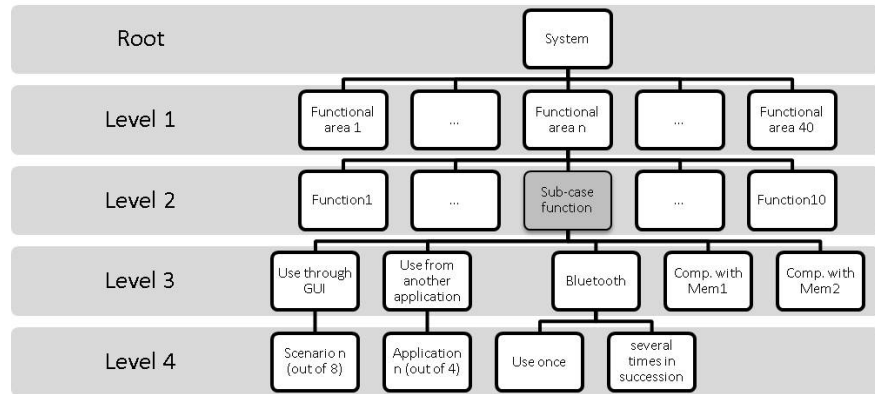


Figure 5: The hierarchical structure of the ‘focus’ parameter in a coverage item. The first level of details contains the divisions of the system in functional areas. Level 2 contains the main functions of a functional area. The ‘sub-case function’ node at level 2 is included in the analysis in this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

The coverage item definitions used for this analysis is not a general proposal 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 4 and the data for pairwise overlay between activities is summarized in Table 5.

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, 90% of the test cases could be considered overlaid since they do not cover any unique coverage items. Out of the test cases explicitly designed at this level 75% are redundant according to our definition (see Section 2.7) In Table 4 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 15 coverage items with 141 test cases at this level. Furthermore the design overlay between test activities at this level is 40%.

Table 4: Coverage data and overlay *within* each test activity for the coverage items. Legend: F = Feature testing, I = Integration testing, S = System testing, Tot = Total, O = Overlay between test activities

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 Cov:I: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%	2%	0%	0%	0%	0%	
	Design redundancy	43%	71%	56%	75%	25%	0%	31%	0%	24%	20%	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%						
	Design redundancy	0%	9%	17%	28%	71%	0%	0%	0%	0%						
	Execution overlay	74%	86%	72%	85%		0%	83%	0%	83%						

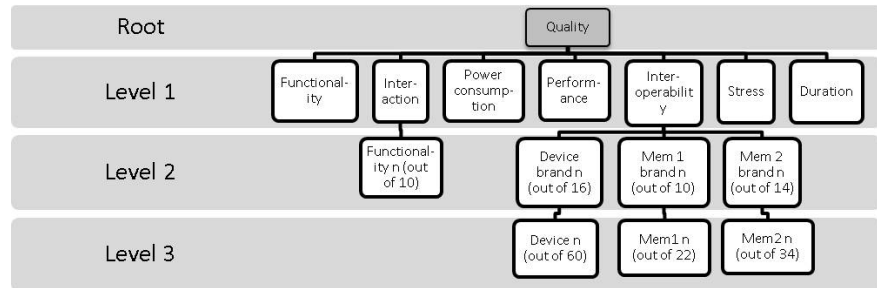


Figure 6: The hierarchical structure of the ‘purpose’ parameter in a coverage item. The first level of abstraction includes the main quality attributes. At level 2 the ‘Interaction’ node is extended with different interacting functionalities and the ‘Interoperability’ node is extended with different 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.

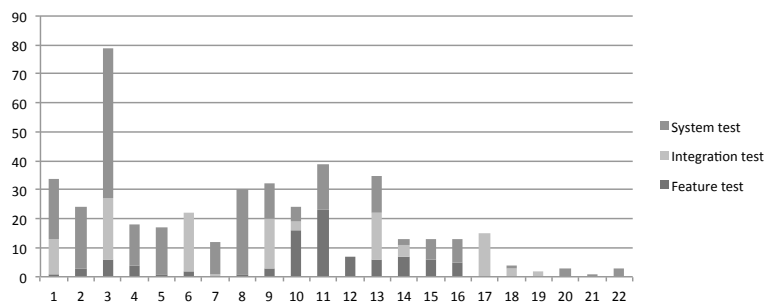


Figure 7: Test executions per test activity and week.

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 activities: integration and system testing (see Table 4), and no overlay within feature testing (see Table 5). There is still design overlay within integration testing and system testing at this level, 33% and 63%, respectively (see Table 4).

Table 5: Pairwise test design overlay *between* test activities 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%

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 or if the delta between the variants have no effect on the 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 at lower abstraction levels indicating a possibility to reduce a large amount of tests due to redundancy.

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.

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 testing carried out is inefficient*, since redundant test cases are not expected to detect new faults (Proposition 2).

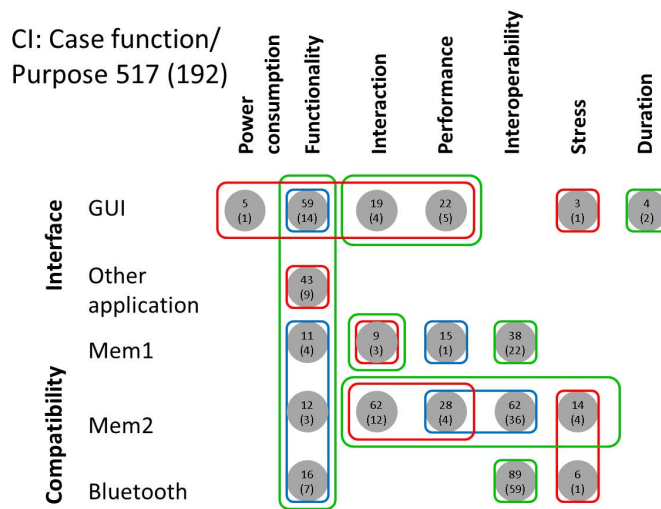


Figure 8: Coverage items covered by the feature testing (blue outline), integration testing (red outline) and product testing (green outline). Numbers within parentheses is the number of designed test cases and the numbers without is the executions.

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 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 test 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 6.

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

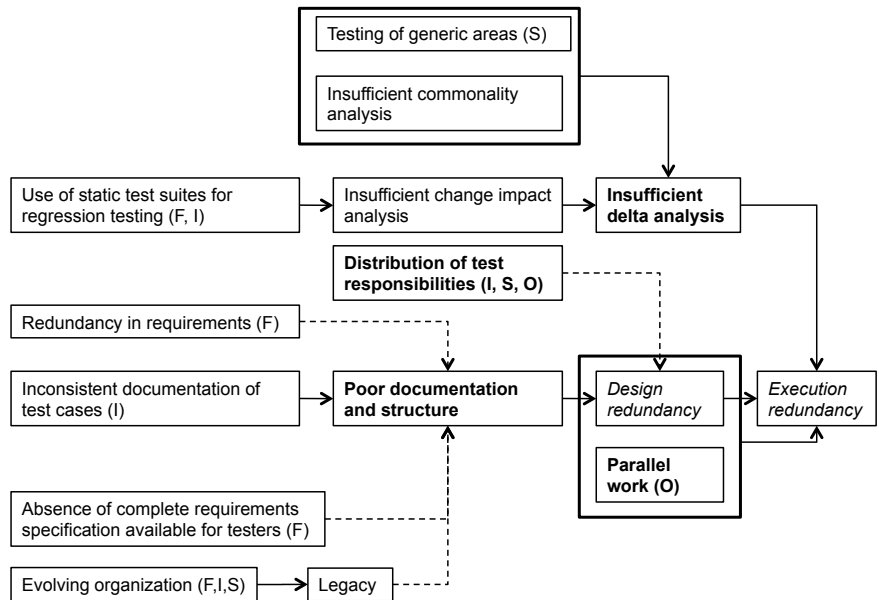


Figure 9: Graph illustrating the hypotheses derived from the qualitative analysis. Text in bold represents our initial propositions. Letters within parentheses denote which units of analysis the hypotheses concern. Arrows A1–13 are linked to the hypotheses in Table 6. Dashed arrows indicate relations partly contradicted by the quantitative data. F = Feature testing, I= Integration testing, S = System testing, O = Overlay between test activities.

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 6, 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 within system testing thanks to a well defined process and good communication (in contrast with H4b and H4c).

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

Table 6: 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 testing carried out is inefficient. (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 activities: 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 activities: 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)

or more chains of arrows within the graph and one arrow may relate to more than one hypothesis.

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 4, there is some variation between abstraction levels as well as between test activities. The overlay within a test activity is more frequent than the overlay between test activities 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 activities. 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 activities but only 25% of the reached coverage.

To test hypothesis H4, the share of test overlay between the test activities is analyzed with respect to the reached coverage. Pairwise overlay is analyzed for three pairs of test activities (feature testing vs. integration testing, integration testing vs. system testing and system test vs. feature testing) and is related to the aggregated coverage of the two overlaid levels analyzed. We also analyze the share of unique coverage in relation to the total coverage within each test activity, see Table 5. Pairwise overlay occurs mainly at the highest abstraction levels i.e. the general test cases overlay between the different test activities while the detailed test cases varies in focus between the test activities. Both integration testing and system testing have 100% unique coverage at their lowest 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 abstraction level P1@F3 the share of unique coverage is low for all three test activities (between 14% and 20%) i.e. most of the covered CovI:s at this abstraction level is covered within another test activity 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-overlapping 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 4. This weak support of H3b is partly explained by the inconsistent document structure, see Section 5.3.

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 4. However if decreasing the level of abstraction the data does not fully support the hypothesis. No internally overlaid (i.e. overlay within a test activity) feature tests exist at this abstraction level (P2@F3). This means that H3a is *not* 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 and 31% is redundant 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 test cases 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.

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 supports that the lack of delta analysis of the customer requirements is a cause of overlay.

Hypotheses H5 and H6 regard overlaid test executions. The share of overlaid test executions is high within all test activities and at all abstraction 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.

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 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): 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 [8, 24]. However, most of these techniques are very context dependent, and in many cases only useful for small systems or small test cases [16].

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), 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 in SPL. 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 SPL context. The SPL 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 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):

1. Distribution of test responsibilities – Organizational distribution had greater impact than geographical.
2. 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.
3. 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):

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

Testing could be more effective by improving delta analyses of the SPL 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] Anneliese A. Andrews, Jeff Offutt, and Roger T. Alexander. Testing web applications by modeling with FSMs. *Software and Systems Modeling*, 4(3):326–345, 2005.
- [2] Ulf Askklund, Lars Bendix, Henrik BÅrnbak Christensen, and Boris Magnusson. The unified extensional versioning model. In *Proceedings of the 9th International Symposium on System Configuration Management (SCM-9)*, pages 100–122, 1999.
- [3] 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.
- [4] Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, Boston, 2001.
- [5] 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.
- [6] Emelie Engström and Per Runeson. Software product line testing—a systematic mapping study. *Information and Software Technology*, 53(1):2–13, 2011.
- [7] 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 (ICST'11)*, pages 367 –376, 2011.
- [8] Emelie Engström, Per Runeson, and Mats Skoglund. A systematic review on regression test selection techniques. *Information and Software Technology*, 52(1):14–30, January 2010.
- [9] 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 (ICST'10)*, pages 75–78, 2010.
- [10] Mary Jean Harrold. Testing: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering, ICSE '00*, pages 61–72, 2000.

- [11] ISO/IEC. 9126-1:2001(e), international standard software engineering product quality part 1: Quality model. technical report. Technical report, 2001.
- [12] Frank J. van der Linden, Klaus Schmid, and Eelco Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer, 1 edition, July 2007.
- [13] Matthew B. Miles and A. M. Huberman. *Qualitative data analysis: an expanded sourcebook*. SAGE, January 1994.
- [14] Kai Petersen and Claes Wohlin. Context in industrial software engineering research. In *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM '09)*, pages 401–404, October 2009.
- [15] Klaus Pohl, Günther Böckle, and Frank J. Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag, 1 edition, September 2005.
- [16] 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 *Proceedings of the 24th International Conference on Software Engineering (ICSE '02)*, pages 130–140, 2002.
- [17] Per Runeson and Emelie Engström. Software product line testing—a 3D regression testing problem. In *Proceedings of the IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST)*, pages 742–746, April 2012.
- [18] Per Runeson, Martin Höst, Austen Rainer, and Björn Regnell. *Case Study Research in Software Engineering—Guidelines and Examples*. Wiley, 2012.
- [19] Graeme Shanks. Guidelines for conducting positivist case study research in information systems. *Australasian Journal of Information Systems*, 10(1):76–85, 2002.
- [20] Antti Tevanlinna. Product family testing with RITA. In *Proceedings of the Eleventh Nordic Workshop on Programming and Software Development Tools and Techniques (NW- PER'2004)*, Turku, Finland, 2004.
- [21] 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.
- [22] 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 *Proceedings of the Third International Conference on Research Challenges in Information Science*, pages 313–324, 2009.
- [23] Eleaine J. Weyuker. Testing component-based software: a cautionary tale. *IEEE Software*, 15(5):54–59, October 1998.
- [24] Shin Yoo and Mark Harman. Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification and Reliability*, 22(2):67–120, March 2012.
- [25] 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.

SUPPORTING TEST SCOPING WITH VISUAL ANALYTICS

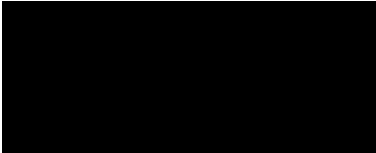
Abstract

Test managers have to repeatedly select test cases for test activities during evolution of large software systems to detect as many faults as possible at a reasonable cost. We explored the use of visual analytics on test data for decision support by creating prototype visualizations and let test managers evaluate them in three focus groups. Our analysis comprises: data extraction, data transformation, visual mapping, user interaction as well as which tasks to support. Although all test managers in the study found the visual analytics useful for supporting test planning, our results show that different tasks and contexts require different types of visualizations. This multitude, together with the many degrees of freedom in transforming data to views and combining data from different sources, makes it challenging to design the best visualization for a specific purpose. Our results may be used to support the introduction of visual test analytics in an organization.

Emelie Engström, Mika Mäntylä, Per Runeson and Markus Borg
Technical report LU-CS-TR: 2013-252

1 Introduction

In the evolution of large software systems, test management has to repeatedly select test cases for test activities to detect as many faults as possible at a reasonable cost. Research on regression testing has introduced several techniques for test selection; we found 28 empirically evaluated techniques [8]. However, most of the techniques do not involve human interaction in the scoping procedure and thus they offer limited support in a complex and variability intensive industrial environment. Furthermore the techniques are only applicable under certain circumstances which could be related to the complexity and type of system under test or to which language software is written in. Similarly, we have searched test scoping techniques



for software product line testing, which means managing parallel variants of products from the same software core in addition to sequentially existing versions [5]. We found the primary challenge to be the large number of potential tests. These observations call for new approaches to test scoping, which handle large number of test cases, are flexible to context variations and allow dynamic interaction with test managers to utilize their experience.

Visual analytics [12] can support any task requiring interplay between humans and computers in order to reason from complex data. It is defined as “*the science of analytical reasoning facilitated by interactive visual interfaces*” [2]. One example of visual analytics of test data is project managers using visualization of failure rates of testing to assess product quality and answer questions like: *What areas need more defect fixing effort?* or: *When can we release the product?* The focus of this study is to use visual analytics for the purpose of improving test quality or planning of testing, for example, answering questions like: *Are there gaps or redundancy in our testing?* or: *What do we need to test due to this change?*

Introducing visual analytics into a context involves four stages of development and validation [10]. 1) The first stage is to characterize the problem and the available data. We have previously studied the test scoping problem through a survey on regression testing practices [4], an action based case study on regression test improvement [7] and an in depth case study on test overlay [6]. This paper builds on our experiences from those studies. The following three stages are more creative and involve three different design problems: 2) to map the problem characterization to an information visualization problem e.g. data types and operations, 3) to design the visual encoding and interaction and 4) to implement this design with an effective algorithm. In this paper, which is addressed to practitioners in the field, we provide advice for the second and third stages based on a case study. We used an existing visualization front end to create hands on prototypes. Validation of its implementation is not within the scope of the paper.

2 Case study

We explored how visual analytics on test data can support test scoping decisions in different industrial contexts. We did this by creating prototype visualizations and evaluate their usefulness in focus group meetings with test managers. We used data from three real-world software contexts and created between three and six different prototype variants per context to focus the evaluation on different aspects of the visual analytics. In addition to the variation in data set, prototypes varied in terms of amount of simultaneous information, how different dimensions of the testing were mapped to the visual structure and whether or not similarities in execution patterns were visualized.

In the focus group meetings, we altered discussions with practical exercises. The practitioners used the prototypes to perform test scoping tasks, T1-4 below,

The empirical study in this article has characteristics of a **case study**, in that it “studies a contemporary phenomenon within its real-life context” (Runeson 2012, p12). However, we isolate a portion of the context, in terms of test data from the three cases, and add the prototype visualizations as an intervention in the study. Further, we use focus groups for data collection (Runeson 2012, p54-55), to make the data collection more efficient, and for the added value of interaction between the study participants. The focus group participants conducted a set of test evaluation and scoping tasks, reported and discussed their findings during the tasks, and the researchers observed their actions and reflections. The primary steps of the procedure are:

- *Design the study* – set up the study goals, collect the test data, pre-process it for use in MosaiCode, define focus group procedures and questions, schedule meetings.
- *Collect data* – run the focus group meetings and have participants do the tasks, catalyze discussions, observe actions, take notes.
- *Analyze data* – collect notes from participants and observers. Three authors independently coded the notes and grouped them into findings, we used these sets of findings to form the conclusions. The fourth author coded the notes based on the merged coding scheme from the first three. The outcome was then used for validation.
- *Report* – write this article.

Reference

Per Runeson, Martin Höst, Austen Rainer, and Björn Regnell. Case Study Research in Software Engineering—Guidelines and Examples. Wiley, 2012.

for specified development scenarios:

- T1 Get an overview of what is included in the test data base. *Is the amount of designed test cases reasonable?*
- T2 Assess test execution progress. *Which areas are well-tested? Which areas need more testing? Is the test coverage sufficient? Explain! Which areas lack test cases?*
- T3 Plan a test session for a new platform (no changes in functionality). *How many/which test cases need to be run?*
- T4 Select and prioritize test cases after a change. *How many/which test cases need to be run?*

We collected feedback on which views and interactions were used, why, how and in which order as well as which were not useful and which were missing.

Table 1 shows an overview of the three data sets used for prototyping visual analytics. In the first focus group meeting, testing consultants from three different organizations evaluated visual analytics of the Firefox data. In the other two meetings, practitioners from organization A and organization B, respectively, evaluated prototypes based on their own data in addition to the Firefox data.

Table 1: The three data sets on which prototypes were created

Case	Organization A	Organization B	Firefox
Characteristics	Emerging software product line	Safety critical software	Open source
Scope	Testing of one single function across organization	7 consecutive executions for a specific release	Full functional test suite for releases 2.0-9.0
Test cases	192	1 059	1 524
Test case executions	517	1 456	272 832
Time period	22 weeks	19 months	6 years

3 Prototype visualizations

As a front end for the visualization, we used the MosaiCode prototype tool [9], which was originally designed for visualizing structure and source code metrics of large scale software, see Figure 1. The tool consists of the primary Mosaic window (1), with structural elements displayed as tiles which are grouped in rectangles (containers). We mapped the test cases or test coverage items¹, TCIs [6]², to these tiles, i.e. each tile represents one test coverage item from one perspective of the testing. Each element is associated with a set of attribute values which may be visualized with different colors for different ranges of values. A tree browser window (2) visualizes the hierarchical structure of the elements and enables selection of elements to study. Below the tree browser there is functionality for searching elements. Finally, a summary window (3) shows a bar chart of the data visualized in the mosaic window. However, the front end is just the visible part of the analytics, which also involves data extraction, data transformations and visual mappings [1].

4 Aspects of visual analytics

This section reports the outcome of our evaluation. We synthesized the feedback from the focus groups, and our experiences from creating the prototypes, according to the different facets of information visualization as described by Card et al. [1]: the task to support with the visualization, the available data, the transformations (from raw data to data tables, to visual structures, to views) and the user interaction in the transformations. A summary is provided in Table 2.

¹Entities describing the scope of a test or set of tests

²In the referenced paper the term used is only coverage items

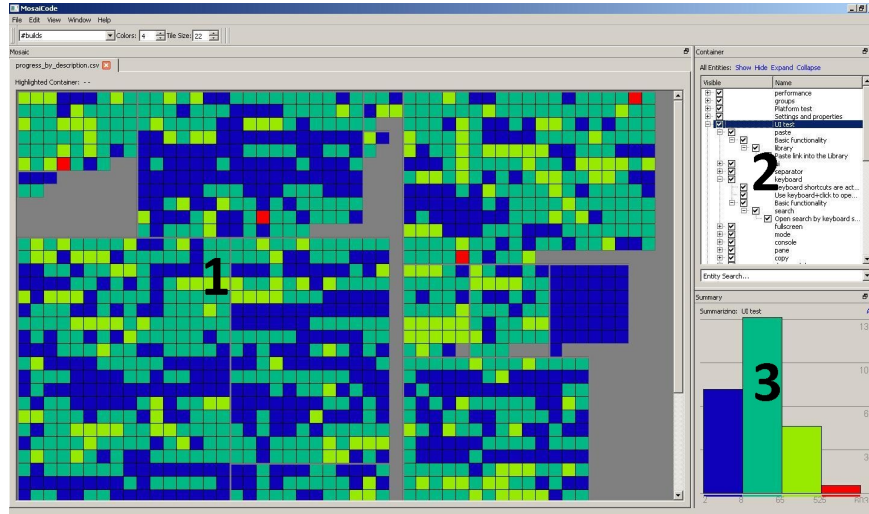


Figure 1: MosaiCode tool visualization [9]

4.1 Test management tasks to support with visual analytics

Visual analytics can support several goals in test management. In a previous case study we identified three types of visualizations that could support test management: visualization of *test design coverage*, *test execution progress* and visualizations of *priorities of tests* related to changes or variability in a system [6]. Visual analytics can also be used for communication with managers: *See these red mosaic tiles have not been executed yet. Are you sure you want to release now?*, or to assign test cases to software testers: *When selecting test cases, keep in mind that our goal for this week is to turn the red mosaic tiles into blue.*

In the focus groups, we asked the participants to perform concrete tasks with the help of our prototype visualizations. The tasks involved getting an *overview* of testing (T1), *assess its sufficiency* (T2) and make decisions about what to test, *plan test*, in two different situations: after a specified functional change (T3) and after porting to a new platform without any changes in functionality (T4). Following is one example workflow based on practitioners' feedback and our observations:³

1. I use the mosaic view where tiles are grouped according to functionality under test to get an overview of the test coverage.

³Statements in this example are not verbatim quotes from the focus groups, rather they represent the synthesis of our intention with the visual analytics, the practitioners' feedback and our observation of how they actually used the prototype visualizations.

2. Depending on the purpose of the specific test session I may assess the sufficiency of coverage.
3. The colors of the tiles help me assess to what extent different parts of the system have been previously tested.
4. To set the scope for a new platform I chose to study the attribute variable: “execution similarity over platforms”.
5. I use few (two or three) tile colors to distinguish different ranges of attribute values.
6. I select a set of test cases with low platform similarity.
7. By changing the limits of the value ranges I may control the size of the selection
8. I also added test cases that failed in the latest release

Based on the hands-on usage of the prototype visualizations, all participants perceived that the visual analytics could be used for many purposes, for example, reducing test resources and evaluating aspects of the testing, but also for generating new test ideas. Following list summarizes the practitioners’ feedback on the usefulness of the visual analytics in relation to different tasks.

- **Getting an overview of testing** – All participants found the visual analytics useful to get an overview and for browsing test data. Some participants also requested mosaic views where other perspectives of the testing were exposed, for example the time, project, or quality dimensions.
- **Assessing test sufficiency** – When assessing test design coverage (T1), the participants saw that test case execution data could provide information about which areas are well covered. To identify areas lacking testing, they wished to combine this information with information from other sources such as customer feedback: “*It is the areas where failures end up in the hands of users.*” or the defect tracking system. While some of this additional information is well documented and maintained in data bases, some is not equally accessible but may exist in distributed reports or informal email conversations. Thus, combining information from several sources in an effective way to make informed test scoping decisions is attributed to the test managers’ experience and expertise. This illustrates the need to incorporate humans in the loop rather than to design a fully automated decision support system, although a visual analytics system may evolve incrementally towards higher levels of automation.
- **Assessing test execution progress** – When assessing test execution progress (T2), participants’ were positive: “*The visualization could be used to show*

everyone's progress." *"The visualization could pinpoint areas to improve during spare time."* In cases where the participants were well familiar with the test scope, the gain in visualization was less *"By experience I know what to keep track of"*.

- **Planning testing** – For both the decision making tasks (T3 and T4), all participants found views that they perceived supportive: *"Deciding on scope the visualization helps identify similarities and to sort out."* *"Good support to make hope informed decision"*

4.2 Test execution history data

We focused on visualizing historical test information and extracted data from the test documentation in the three organizations. In the Firefox case, we extracted test data from their open, web-based, test management system, Litmus, with a web crawler. In organization A data are stored in a similar commercial system, HP Quality Center, from which we manually extracted test information. In organization B, test executions were documented in MS Word documents and stored in a general document management system. We extracted this data (semi-) automatically.

The focus group participants perceived that visualization of test execution history was useful for assessing test sufficiency and for making decisions. As stated above, some participants also suggested that the visual analytics tool should collect data from additional data sources such as the source code repository: *"We need a link to software delta"*, requirement management system *"Link to requirements would be powerful as well."* and the defect tracking system. Such a system would have similarities with project intelligence platforms which enable trend and comparative analyses by collecting process and product metrics from several sources [3]. However, a prerequisite for enabling such analyses is that the information is available, and that organizations have policies allowing information integration. Unfortunately, large organizations often manage information in different content management systems, typically offering limited interoperability (i.e. resulting in "information silos") [11].

4.3 Dimensions and attributes of the test coverage items

In our visualization proposal, we transform test execution information to data tables or "cases by variable arrays" [1] where test coverage items (TCIs) are the cases. A TCI describes the scope of a test or set of tests [6] and is defined by a set of variable values. We differ between two types of variables: *dimension* variables with textual values and *attribute* variables with numerical values. A dimension value describes the hierarchical position of the TCI from one perspective, for example:

Dimension: Functionality = System/Social phonebook/Add contact

or

Dimension: Variant = All/Blue star/US-market

TCIs may be defined at different levels of abstractions depending on test strategy, for example, when exploratory testing is applied, TCIs are defined at a high abstraction level while scripted testing enables finer analysis. The abstraction level of the TCI determines the derived values of the attributes which could be for example:

Attribute: Number of executions = 13

or

Attribute: Number of days since last execution = 30

There is no limitation in how many variables to include in the analysis. However, the number of TCIs increases exponentially with the number of dimension variables. In total, we visualized five different dimension variables: *project* and *functionality*, for all data sets; *execution patterns*, for the Firefox and organization B data; *platform variants*, for the Firefox and organization A data; and *process* for the organization A data. Some dimension values we could extract directly from the documentation, such as project information (in terms of release numbers), platform variants and process information (in terms of test activities). To derive the hierarchical structure of the functionality dimension we had to interpret the test case descriptions in the light of available documentation and expertise. In the Firefox and organization B cases we based this classification on keywords, while in the organization A case we did it manually. The *execution patterns* dimension refers to a classification based on similarities between test cases with respect to when, on what and with which verdict they have been executed.

Dimensions – Our evaluation revealed needs to visualize different dimensions of the testing, depending on the user’s role and tasks. In the organization A focus group, one of the participants suggested an organizational dimension to evaluate the testing by teams or even by individuals. The functionality dimension was useful to get an overview of the scope and to navigate the information. The classification based on similarities in execution patterns were appreciated for planning but was not intuitive and thus required certain user skills. Preferably, several dimensions should be available at the same time, for example with multiple tabs. In our prototype visualizations we did not make use of the two-dimensional positioning in the mosaic window, which could enable pairwise overviews of TCI dimensions, for example functionality versus different quality attributes. Instead we created a two-dimensional overview by mapping the dimension values of one dimension variable to a set of attribute variables.

Attributes – Our evaluation also showed the need to measure different attributes of the TCIs for different tasks. *Failure rate* showed to be useful for evaluating the test design coverage. Several attributes were considered by at least one of the participants when evaluating execution progress: *#builds tested*, *#languages tested*, *#platforms tested*, *#days since last execution*, *#executions*, *#failures*, *failure rate* and *#days since first execution*. In addition to these attributes they used measurements of *similarity of execution patterns* over different platforms to plan testing of new product variants. It is important that the naming or description of attributes are clear and thus consistently understood by all users of the tool.

4.4 The visualization front end

The main components of a visual structure are the use of space, the marks and the graphical properties of the marks [1]. In the prototype visualizations we used spatial position to encode the dimension values of the TCIs in the mosaic window. The MosaiCode tool [9] only supported unstructured type of axis [1] in the mosaic window. This means that the coordinates of the tiles had no meaning while the arrangement of tiles into containers carried meaning. Thus, the dimension values, or the hierarchal position, are visible to the user through the grouping of tiles and in the tree browser, see Figure 1. The attribute values are visible in the colors of the mosaic tiles and in the bar chart view on the right.

This lack of a coordinate system in the mosaic window was frustrating for the testers. Participants in all focus groups requested at least a fixed position of tiles. They also suggested that the meaning of the containers should be visible, not only in the tree browser window, but also in the mosaic window. This could be done by adding text labels to the containers or to nominal axes. Participants also commented on the number of colors used to represent different ranges of values of the attributes. The general opinion was that only few colors should be used; two or three would be enough.

In the evaluation, we studied how the participants used the different views to solve the tasks. The tree browser view showed to be useful for navigating when participants were familiar with the test scope and the structure of TCIs. The mosaic view provides a good overview of both TCI design coverage and TCI execution progress and showed to be useful when they were new to the data or when the data set was large.

Table 2: Overview of the facets of visual test analytics (tasks, raw data, data transformation, visual structures and human interaction) evaluated in our study. The columns show how and if it was implemented in a prototype and the main observations and suggestions from the focus group discussions.

Facets	Implemented in prototypes	Observations	Suggestions	
Tasks	Overview test cases	Visual analytics was useful		
	Assess test design coverage	Performed in focus groups	Visual analytics was useful to identify well tested areas	
	Assess test execution progress		Visual analytics was useful	
	Plan testing after change			
	Plan testing for new platform			
Raw Data	Test execution history	Test reports from test management data bases and MS word documents	Visual analytics was good to overview large amounts of data	
	Change impact analyses	Not in prototype	Would improve test planning support	
	System information		Would support assessment of test design coverage	
Data Transformation	TCIs	Test cases	Easily extracted from existing documentation	
		Implicit testing goals	Requires more effort, reveals test overlay in complex organizations	
	Dimension variables of TCIs	Project	Good for assessing progress	Visualize organizational dimension to evaluate teams or individuals. Visualize time dimension to assess progress. Visualize multiple dimensions of the TCIs
		Functionality	Good to get overview of scope	
		Execution pattern	Good for planning, less intuitive	
		Platform variants	Good for planning	
	Attribute variables of TCIs	Similarity in execution patterns over different platforms	Useful for planning testing on new platform	
#Executions, #Failures, #Builds, #Languages, #Platforms, #Days since first execution, #Days since last execution		Useful for planning and assessing execution progress		

continued on next page ...

Table 2 – continued from previous page

	Facets	Implemented in prototypes	Observations	Suggestions	
Visual Structures		failure rate	Useful for all tasks		
	Marks	TCI:s mapped to Mosaic tiles	Provides a good overview of both		
	Use of space	Arranged in containers	test design coverage and TCI execution progress, especially when data set is large	Add text labels on containers	
		Positioned in unstructured coordinate system		Maintain spatial position, Map TCI dimensions to nominal axes	
	Graphical properties of marks	Attribute values mapped to colors	Only few colors are needed		
Additional views	Dimension values mapped to tree structure	Useful for navigating when familiar with scope			
	Summary in bar chart	Useful for planning			
Human Interaction	Interactive data transformation	Not in prototype	Human interaction is important	Enable manual filtering of data	
	Interactive visual mapping			Enable manual mapping of TCI dimension to spatial position	
					Enable manual mapping of attribute values to colors
	Interactive view transformation	Control of tile size		Prefer automatic setting	Enable selection and counting of test cases to support test planning
		Control number of colors			
Search			Useful for planning testing after change		
Browsing					
	Zooming		Very useful		

4.5 Human interaction

Participants in all three focus groups stressed the importance of human interaction. Several techniques exist to let the user control the different types of data transformations in the visual analytics process [1]. In the prototype visualizations, the *transformation from raw data into data tables* as well as the *mapping from data tables to visual structures* was hard coded. Thus, the user interaction provided in focus groups, with respect to these transformations, was limited to selecting which prototype visualization to use for a certain task. Participants in all focus groups requested a possibility to filter the displayed data on different criteria, e.g. visualize test execution history of functional area F by team A from Q4.

Participants discussed two types of interactive visual mapping which they considered useful namely 1) the selection of which TCI dimension to be mapped to

the spatial position in the mosaic view and 2) the setting of percentiles in the bar chart, that is, the mapping of colors to attribute values.

The MosaiCode tool provides *interactive view transformations* in terms of browsing and zooming as well as controlling the tile size and number of different colors on tiles. In addition, automatic mapping between the different views enables searching for elements in, for example, the tree browser by selecting them in the mosaic window. Practitioners considered the zooming function very useful and stressed the importance of a direct mapping between different views i.e. if a certain test coverage item is selected in the tree browser view the same item should be selected in the mosaic view. Furthermore they requested automatic setting of tile sizes but manual setting of tile color ranges based on the TCI attributes. Additionally, they wanted to be able to select and count test cases to support the test planning.

5 Conclusion

Table 2 provides an overview of the results from our case study. Five main facets of visual analytics were analyzed: 1) the tasks to support, 2) the raw data to use, 3) the transformation to data structures, 4) the mapping to visual structures and 5) the human interaction. The second left column contains the topics analyzed while the two rightmost columns contain observations and suggestions to guide the implementation of visual analytics.

All study participants confirmed potential usefulness of the visual analytics for test scoping as well as for communicating decisions with managers and subordinate testers. Tasks to support relate to the assessment of test design coverage and test execution progress as well as to the planning of testing after a change or for a new platform. Visual analytics seem to be especially beneficial if it enables aggregation of all test execution data across the organization as well as the parallel and consecutive projects, combined with the ability to filter the displayed data. Test execution data may be transformed to sets of dimensions and attributes, TCI:s, where the dimension variables represent different perspectives of the testing and the attribute variables represent different properties of the TCI:s. The mapping of dimensions variables to the spatial position is important but not obvious. Different tasks require views from different perspectives. Testing goals and strategies include several dimensions of variability. Thus a visualization tool needs to be flexible and allow multiple perspectives on the testing.

Organization A develops mobile devices based on the Android platform which comprises more than 10 million lines of code. The development context is variability intensive i.e. a software project comprises about 10 different product variants, instances of specific combinations of hardware and software, which in turn are customized for hundreds of different pairs of customers and market segments. The development organization is globally distributed over three continents. Testing tasks are organizationally distributed over three main units: core software, application software and product composition. Organization A applies incremental development practices which imply a need for continuous regression testing at all levels of test.

The four participants in the organization A focus group represent different levels of test and organizational units. All participants analyze test results in their work but with different purposes, e.g. test management in specific development projects, general test process improvement, line management and product quality assessment.

Mozilla foundation develops the **Firefox** web-browser. Currently, Firefox has roughly 10 million lines of code and the typical number of people committing code at least once monthly is over 200. Developers perform lower level testing, and developers and feature managers test that new features have been implemented correctly and large crowds participate in the alpha and beta-testing of the upcoming product releases. Main regression testing involves about 30 active individuals frequently reporting test results to the open test management data base. Testing is done mainly for three major platforms: Windows, MacOSX and Linux.

In the mixed focus group, three software testing consultants from three different organizations studied the visualizations of test execution data from the Firefox projects. The participants had worked in various roles, e.g. a test manager, test competence manager, coach of software testing, software test and process auditor, and project manager. They all had hands on as well management experience in software testing. The least experienced participant had five years of software testing experience while the most experienced had over ten years.

Organization B is a large multinational company operating in robotics and in the power and automation sector. The studied development context comprises safety-critical embedded development in the domain of industrial control systems. The development site adheres to the safety standard IEC 61511, and the product is certified to a Safety Integrity Level of 2 as defined by IEC 61508. The number of developers is in the magnitude of hundreds; a project has typically a length of 12-18 months and follows an iterative stage-gate project management model. The verification process covers software testing on several different abstraction levels, partly conducted by independent testers. Furthermore, the process includes a partly automated suite of test cases that are executed daily, as well as a suite of regression tests that are executed prior to releasing new versions.

In the organization B focus group, a test manager and a project manager participated. Both participants had experience of planning test activities, including resource allocation and prioritization. Two central work tasks for test management relate to test selection. First, managers define a test scope for new releases, which depends on how the system has changed. Second, they specify which test cases should be included in the formal regression test suite. Moreover, as an external agency certifies the selection, it is less feasible to change. Finally, following up test progress is critical. Currently, graphs based on earned value management visualize test progress.

Acknowledgement

The authors are very thankful to the case companies and focus group participants, for letting us access them and their data, as well as giving valuable feedback on our findings. We also thank the SERG reading group for valuable review comments to an earlier version of the paper. This work was partly funded by ELLIIT (The Linköping-Lund Initiative on IT and Mobile Communication, www.elliit.liu.se) and EASE (Industrial Excellence Center on Embedded Applications Software Engineering, ease.cs.lth.se). Emelie Engström and Markus Borg are members of the SWELL research school, (Swedish V&V Excellence, www.swell.se).

References

- [1] Stuart K. Card, Jock Mackinlay, and Ben Shneiderman, editors. *Readings in Information Visualization: Using Vision to Think*. Academic Press, 1 edition, February 1999.
- [2] Kristin A. Cook and James J. Thomas. Illuminating the path: The research and development agenda for visual analytics. Technical Report PNNL-SA-45230, Pacific Northwest National Laboratory (PNNL), Richland, WA (US), May 2005.
- [3] Florian Deissenboeck, Elmar Juergens, Benjamin Hummel, Stefan Wagner, Benedikt Mas y Parareda, and Markus Pizka. Tool support for continuous quality control. *IEEE Software*, 25(5):60–67, October 2008.
- [4] 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.
- [5] Emelie Engström and Per Runeson. Software product line testing—a systematic mapping study. *Information and Software Technology*, 53(1):2–13, 2011.
- [6] Emelie Engström and Per Runeson. Test overlay in an emerging software product line—an industrial case study. *Information and Software Technology*, 55(3):581–594, March 2013.
- [7] 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 (ICST'11)*, pages 367–376, 2011.
- [8] Emelie Engström, Per Runeson, and Mats Skoglund. A systematic review on regression test selection techniques. *Information and Software Technology*, 52(1):14–30, January 2010.
- [9] Jonathan I Maletic, Daniel J. Mosora, Christian D. Newman, Michael L. Colvard, Andrew Sutton, and Brian P. Robinson. MosaiCode: visualizing large scale software: A tool demonstration. pages 1–4, September 2011.
- [10] Tamara Munzner. A nested model for visualization design and validation. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):921–928, December 2009.

- [11] Jamshid A. Vayghan, Steven M. Garfinkle, Christian Walenta, Donald C. Healy, and Zulma Valentin. The internal information transformation of IBM. *IBM Systems Journal*, 46(4):669–684, 2007.
- [12] Pak Chung Wong and J. Thomas. Visual analytics. *IEEE Computer Graphics and Applications*, 24(5):20–21, October 2004.