LUND UNIVERSITY

Exploring regression testing and software product line testing - research and state of practice

Engström, Emelie

2010

Exploring Regression Testing and Software Product Line Testing - Research and State of Practice

Emelie Engström

Licentiate Thesis, 2010

Department of Computer Science
Lund University
Faculty of Engineering

ISSN 1652-4691
Licentiate Thesis 11, 2010

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

Email: emelie.engstrom@cs.lth.se

Abstract

In large software organisations with a product line development approach a selective testing of product variants is necessary in order to keep pace with the decreased development time for new products, enabled by the systematic reuse. The close relationship between products in product line indicates an option to reduce the testing effort due to redundancy. In many cases test selection is performed manually, based on test leaders' expertise. This makes the cost and quality of the testing highly dependent on the skills and experience of the test leaders. There is a need in industry for systematic approaches to test selection.

The goal of our research is to improve the control of the testing and reduce the amount of redundant testing in the product line context by applying regression test selection strategies. In this thesis, the state of art of regression testing and software product line testing are explored. Two extensive systematic reviews are conducted as well as an industrial survey of regression testing state of practice and an industrial evaluation of a pragmatic regression test selection strategy.

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. Several techniques for regression test selection are proposed and evaluated empirically but in many cases the context is too specific for a technique to be easily applied directly by software developers. In order to improve the possibility for generalising empirical results on regression test selection, guidelines for reporting the testing context are discussed in this thesis.

Software product line testing is a relatively new research area. The understanding about challenges is well established but when looking for solutions to these challenges, we mostly find proposals, and empirical evaluations are sparse. Regression test selection strategies proposed in literature are not easily applicable in the product line context. Instead, control may be increased by increased visibility of the effects of testing and proper measurements of software quality. Focus of our future work will be on how to guide the planning and assessment of regression testing activities in large, complex reuse based systems, by visualizing the quality achieved in different parts of the system and evaluating the effects of different selection strategies when applied in various regression testing situations.

Acknowledgements

This work was financially supported by the Swedish Governmental Agency for Innovation Systems under Grant 2005-02483 for the UPPREPA project.

First of all I would like to thank my supervisor Prof. Per Runeson for excellent guidance. Thanks for encouraging and challenging me and also for good cooperation. I would also like to thank my other colleagues at the department of computer science and especially the software engineering research group for the inspiring and developing work environment. I am grateful for all persons from industry who have contributed to this work. Thanks to all who have participated in focus group meeting and questionnaires and a special thanks to Dr. Greger Wikstrand at KnowIT YAHM Sweden AB for your enthusiasm and your commitment to our cooperation. Thanks also to other fellow researchers; co-authors, participants in the EASE project and in the SWELL research school. Finally, thanks to my family; to my parents for constant encouragement and for making me believe that anything is possible. And last but not least I'm so grateful for my children David, Miriam and Anton, you teach me what is important in life, and my husband Jonas, thanks for your continuous support and for always standing by my side.

Content

Introduction

1 Introduction

Software product line engineering is a means for organizations to customize large numbers of software products from a common base instead of developing one-off solutions to each customer or end product. Efficient testing strategies are important for any organization with a large share of their cost 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.

Software product line testing is a complex and costly task due to the variability preserved in the product platform and the large number of different products derived from the same platform. The major challenge with SPL testing regards the large number of required tests (Engström and Runeson *in press*). 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. This work aims at developing and evaluating strategies for selective testing of software product line applications in large scale software development organisations in order to minimize the amount of redundant testing in such a context.

This problem is closely related to the problem of regression testing of evolving software in general. The goal of regression testing is to verify that previously working software still works after a change (IEEE Std 1990). The test scope for

regression testing is often set by selecting test cases from an existing test pool, based on knowledge about changes between the system under test and previously tested versions of the system (Engström and Runeson 2010). This could be compared with the testing of a new product configuration in a software product line where the previously tested product line is the older stable version of the system. Our starting point has been the state of art of regression testing since this activity is researched and practiced to a greater extent.

This work surveys existing research and state of practice on regression testing as well as research on SPL testing. 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 (Engström and Runeson 2010). Several techniques for regression test selection are proposed and evaluated empirically but in many cases the study context is too specific for the technique to be easily applied directly by software developers in another context. Few studies are replicated, and thus the possibility to draw conclusions based on variations in test context is limited (Engström et al. 2010a). 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. Guidelines for reporting empirical results on regression test selection are discussed (Runeson et al. 2008) and an evaluation of a pragmatic strategy for regression test selection is provided (Engström et al. 2010b). Software product line testing is a relatively new research area. The understanding about challenges is well established but when looking for solutions to these challenges, we mostly find proposals, and empirical evaluations are sparse (Engström and Runeson *in press*).

This thesis includes five papers and is organised as follows:

- **Introduction.** The introduction gives a background to the research and an overview of the contributions of this thesis. Section 2 provides clarifications on important concepts referred to in this thesis. In section 3 the overall goals and research questions are stated. Section 4 provides an overview of related work on software product line engineering and regression testing. Section 5 describes the research methods used and section 6 overviews the results. Section 7 concludes and discusses directions for our future work.
- **Regression testing in the literature.** This part includes two papers regarding the evidence base on regression testing. Paper 1 reports on a systematic review of techniques for regression test selection and paper 2 is a position paper elaborating on test benchmarks and proposing improvements regarding the analytical generalisation of research on testing.
- **Regression testing in practise.** This part includes two papers exploring and evaluating regression testing practices in industry. Paper 3 reports on a survey, including 46 practitioners from 38 companies, of regression testing practices. Paper 4 reports on the results of an empirical evaluation of a pragmatic approach to regression test selection in an industrial setting.

- **Software product line testing in the literature.** The last part consists of one paper (paper 5) and investigates the state of art of research on software product line testing. It reports on a systematic mapping of relevant research and provides a multidimensional classification of the research contributions.

Included papers

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

2. **Test Benchmarks - what is the question?**
   Per Runeson, Mats Skoglund and Emelie Engström[2]
   Proceedings of *International Conference on Software Testing Verification and Validation Workshop* (ICSTW '08), April 2008

3. **A Qualitative Survey of Regression Testing Practices**
   Emelie Engström[1] and Per Runeson
   Accepted for publication in Proceedings of *11th International Conference on Product Focused Software Development and Process Improvement* (PROFES '10), June 2010

4. **An Empirical Evaluation of Regression Testing Based on Fix-cache Recommendations**
   Emelie Engström[3], Per Runeson and Greger Wikstrand
   Proceedings of *International Conference on Software Testing Verification and Validation* (ICST '10), April 2010

5. **Software Product Line Testing - A Systematic Mapping Study**
   Emelie Engström[1] and Per Runeson
   Conditionally accepted for publication in the journal of *Information and Software Technology*

[1] The author of this thesis is the main author and as such responsible for running the research, dividing the work between coauthors and conducting most of the writing.

[2] The main contribution of the author is indirect as the proposals are partly based on data collection and analysis made by the author.

[3] The author of this thesis is the main author and as such responsible for design, data collection and analysis and division of work between coauthors.

2 Concepts

In this section we provide background information on software product line engineering, regression testing and some general testing concepts which are frequently referred to in this thesis. Since some important concepts may be interpreted differently in different contexts their interpretation within this thesis is clarified as they are introduced here.

2.1 Software product line engineering

In software product line engineering (SPLE) mass customization is achieved through systematic reuse of artefacts throughout the development process. *Commonality* and *variability* are identified at an early stage and the software process is divided into two separate processes: 1) *domain* (platform) engineering and 2) *application* (product, product configuration or variant) engineering. Domain engineering is focused on establishing the reusable platform while application engineering is focused on deriving product line applications from the platform. A large part of application engineering consists of reusing the platform and binding the variability as required for the different applications. A *variation point* is a representation of a variable item or property. (Pohl et al 2005)

2.2 Regression testing

Regression testing involves repetitive tests and aims to verify that previously working software still works after changes to other parts. 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* (IEEE Std 1990). Focus can be either re-execution of test cases or retest of functionality. As for testing in general (Burnstein 2003) the goal of the regression testing may differ between different organizations or parts of an organization. The goal may be either to reveal defects or to obtain a measure of the quality of the system (paper 3). Regression testing is the most expensive test activity and may consume as much as 80% of the testing budget. (Chittimalli and Harrold 2009)

Traditionally regression testing is iteratively applied to the consecutive *revisions* of the software, where revisions refer to the variation in time (Bendix 1995) of the evolving system. In a software product line a counterpart to these revisions are the product line *variants* referring to the variation in space (Bendix 1995) i.e. the different product line configurations. The *delta* between two revisions or variants refers to the difference between them. Although many reports on SPL testing

have raised the possibility of using regression testing techniques for testing variants in a software product line, there are few published reports on how this should be done (paper 5). Of course there is also a need for regression testing in a traditional sense even in the product line context, in order to test revisions of product line variants and of the product line itself.

2.3 Levels of testing

Testing practices and goals differs depending on the levels of abstraction on which it is carried out (Burnstein 2003). At a low level of abstraction, such as unit testing, the goal may be to reveal structural and functional defects in a component and test cases could be designed with both white box and black box techniques. At a system level the behaviour of the system as a whole is tested with various black box strategies, e.g. functional testing and stress testing, both high level functional requirements and quality aspects, e.g. reliability, usability, and performance, may be evaluated. Another common testing level is the integration testing level where the interactions between the components or subsystems are tested. Regression testing appears at all levels of testing.

Depending on the size of the project and the process model followed, the number of abstraction levels on which testing is applied varies. Most commonly referred to are *unit testing*, *integration testing* and *system testing*. These levels derive from the V-model but are often mapped to other process models, e.g. iterative development processes. In a software product line where the process is ideally divided into the domain engineering process and the application engineering processes it is not clear how such mapping should be done.

2.4 Test coverage

Test coverage is a measure of how well a system is covered by tests. The system may be defined by different types of artefacts e.g. requirements, design or code. The definition of test coverage could be based on any model of the system and be measured at any level of abstraction. In the regression test selection context test coverage relates to the execution of tests rather than the design.

3 Research goals and questions

The application of software product line development strategies enables mass customization of complex software systems. A major challenge with SPL testing, or any large, highly configurable software system, 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 configurations, need to be

tested. Such thorough testing is infeasible and a selective testing is necessary. As products of a product line are closely related and derive from the same specifications, it is likely that a large amount of testing, especially in the application development process, can be removed due to redundancy. It is not clear, however, how to make the test selection at different stages of the development process and how to balance the test effort between different test activities. The overall goal of this research is to:

Evaluate possible strategies for improved test selection, aiming at minimizing the amount of redundant testing, in software product line development.

This goal, minimizing redundant testing, is the same as for regression test selection. By focusing the testing on changes to the system and parts affected by those changes regression testing aims at verifying that previously working software still works after a change (IEEE Std 1990). With the assumption that changes in an evolving system could be compared to the differences between applications in a software product line and that strategies for regression test selection may be applied for the purpose of planning the testing of applications derived from software product lines our starting point is the state of art of regression testing. Even though the application of regression test selection strategies for this purpose is frequently suggested (McGregor 2001) (Tevanlinna et al. 2004), few concrete proposals exist.

In this thesis the following five main questions are investigated:

RQ1 What do we know about regression test selection?
In order to identify relevant strategies we want to get an overview of the research and aggregate existing empirical evidence on regression test selection. No single solution to regression test selection could possibly fit into all situations and no single study evaluates every aspect of the regression test selection problem. Limited sets of regression test selection techniques have been compared in previous reviews of regression test selection with different foci (Rothermel and Harrold 1996), (Do et al. 2005) (Juristo et al. 2006) we want to get a more complete picture of the state of art. A systematic review was launched for this purpose (paper 1)

RQ2 What do we mean by an effective test selection strategy?
In order to compare strategies for regression test selection and select the one most appropriate for our purpose a number of questions need to be answered: Do we have a common understanding of what an effective regression test selection technique is; should it be safe or cheap? How do we know which technique is the most appropriate for a specific situation? Given a technique evaluated in a specific context what does the result mean in another context? Which techniques are relevant to compare and with respect to what? This question is partly evaluated by the systematic

review (paper1) and the industrial survey (paper 3). It is also in focus of our benchmark proposal (paper 2).

RQ3 How is regression testing applied in industry?
Regression testing is a frequent and time consuming test activity in most industrial software projects (Chittimalli and Harrold 2009). It is applied in many different contexts, not based on research but based on experience. We want to survey regression testing practices in industry (paper 3) in order to extend the overview from the literature review with experiences from industry. What are the challenges and best practices in industry what is needed to over bridge the gap between research and practice?

RQ4 Can an automatically derived regression test suite be more effective than a manual experience based selection in a large scale industrial context?
Regression test selection is to a great extent performed manually in industry with more or less transparent strategies. Non-systematic manual strategies are heavily dependant on the skills of the test engineers and their experience of the test area. Such strategies are vulnerable to the mobility of people and to time pressure of the projects as it is easier to select the same tests every time than to redo the analysis. They may also be unnecessary expensive since people tend to add some extra test cases just to gain confidence in the testing. Reports on systematic approaches applied and evaluated in a large scale industrial context is however sparse (paper 1). It is a complex task and many proposed strategies are not feasible to scale up to large complex systems We want to apply systematic approaches to regression test selection in industrial contexts and evaluate them against the strategies already in use. One such evaluation is made through launching a post-hoc case study comparing the fix-cache selection strategy with the current strategy at the case company (paper 4).

RQ5 What do we know about software product line testing?
In order to position our research, we also want to get an overview of the current status of the research on software product line testing. What are the challenges? Which topics for testing product lines have been investigated and to what extent? For this purpose a systematic mapping study was launched. (paper 5)

4 Related work

This section contains a brief overview of the related work on software product line testing and regression test selection. Extensive literature reviews of the research in these areas are provided in papers 1 and 5, respectively. More details on how each research question relates to previous research can be found in the

respective papers included in this thesis. Positioning of future work can be found in section 7, conclusions and future work, of this introduction

4.1 Software product line testing

Research on SPLE started in the 90ies and has gained growing interest the last 10 years. Two conference series started with product lines or product families in focus. The main conference series; Software Product Line Conference (SPLC) and Product Family Engineering (PFE) conference started 2000 and 2001 respectively and merged 2005. Main benefits of applying SPLE reported are reduced time to market, reduced cost and improved quality (Khurum and Gorschek 2009)

Testing is still a bottleneck though. Software product line testing has shown to be a complex task introducing many new challenges (paper 5). A thorough report on techniques and activities for meeting those challenges is a technical report by McGregor (McGregor 2001). This work is the starting point for many researchers within the field. The field of software product line testing is immature though and there is a lack of empirical evidence in literature as well as of good practices in industry.

Even though new testing challenges arise within the product line context most traditional testing practices are still valid. Practices for testing object oriented systems could for example be applied to the testing of a software product line as well (Tevanlinna et al. 2004). Some traditional testing strategies may be particularly advantageous in, and adapted to, the software product line context such as regression test selection (Tevanlinna et al. 2004) and model based testing (Olimpiew and Gomaa 2005)(Reis et al. 2007). An overview of challenges and existing research is given in (paper 5)

4.2 Regression test selection

Research on regression testing has been going on for a while; empirical studies are reported on since 1980 and the field is one of the more mature in software engineering. The main focus has been how to select tests based on information about changes in the system since the latest tested version (Hartmann and Robson 1990), (Agrawal et al. 1993), (Chen et al. 1994), (Gupta et al. 1996), (Rothermel and Harrold 1997). Also the prioritization of test cases in a regression test suite has been in focus (Elbaum et al. 2002)(Kim and Porter 2002). Other researched aspects of regression testing are for example; how to evaluate selection techniques (Rothermel and Harrold 1996), how to regression test GUI:s (Memon 2004), and how to regression test databases (Haftmann et al. 2007).

An overview of research on regression test selection is given in (paper 1). Most of the research is conducted as experiments or small scale case studies and one of the challenges is to scale up solutions and apply them in different industrial contexts. Experiments have been undertaken to study the implications of systems of different sizes on regression testing techniques (Bible et al. 2001), (Graves et al. 2001) and (Orso et al. 2004) and a few large scale case studies have been undertaken (White and Robinson 2004) and (Skoglund and Runeson 2005). Another challenge, which is common for all research in software engineering, is how to generalize results and benchmark solutions (paper 2). Rothermel and Harrold (1996) proposed a framework for evaluating regression test techniques which have been used in some proceeding studies, e.g. (Bible et al. 2001) and (Briand 2002).

5 Research methods

5.1 Exploratory research

Initially the research has mainly been exploratory (Easterbrook 2008). In order to address the first and the fifth research questions, two different kinds of systematic literature reviews (SLR:s) have been conducted: a systematic review on regression test selection (paper 1) and a systematic mapping on software product line testing (paper 5). 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 (Brereton et al 2007), (Kitchenham et al 2007), (Staples and Niazi 2007) and updated accordingly (Kitchenham 2007). Kitchenham et al. (2009) recently published a review of 20 systematic reviews in software engineering during 2004–2007.

A mapping study is an alternative to systematic reviews and could be used if the amount of empirical evidence is too little, which was the case for the software product line testing, or if the topic is too broad for a systematic review to be feasible. Both methods are systematic in that a well defined protocol for study selection and analysis is followed but the goal and use differs. A mapping study is performed at a higher granularity level than a systematic review, aiming at identifying research gaps and clusters of evidence in order to direct future research. The goal of a systematic review is to analyse and aggregate the base of empirical evidence. Petersen et al. (2008) describe how to conduct a systematic mapping study. Guidelines for performing SLR:s can be found in (Kitchenham 2007). The second question is elaborated on through reasoning about observations in the first systematic review.

In order to address the third research question, a qualitative survey has been conducted (paper 3) by means of focus group discussions and a questionnaire to validate the results. Survey research is traditionally used to identify the characteristics of a broad population of individuals and the most common use of survey research is when questions are quantitative e.g. to what extent do developers use this tool and how satisfied are they? (Easterbrook 2008) One defining characteristic of survey research is that a representative selection should be done from a well-defined population. In this case it was not possible or necessary, since our conclusions are qualitative rather than quantitative, Instead the sample is selected by availability and interest. In total 38 software organizations were represented by 46 testers in the survey.

5.2 Evaluation research

In addition to the exploratory studies an evaluative study in the form of a case study has been conducted (paper 4), which partly answers question four. We have evaluated a pragmatic strategy; Fix-cache based regression testing, for improving efficiency of regression testing in a large industrial setting. This case study was applied post hoc. Two different selection strategies were applied to a large scale industrial project and data collection and analysis was made afterwards.

Table 1 shows the relations between papers included in this thesis, the research method used and the research questions addressed.

Table 1. Relations between papers, research methods and research questions

Paper	Type of research	Research method	Research Question
P1	Exploratory	Systematic Review	RQ1, RQ2
P2	Exploratory	Proposal	RQ2
P3	Exploratory	Survey	RQ2, RQ3
P4	Evaluative	Case Study	RQ4
P5	Exploratory	Systematic Mapping	RQ5

6 Contributions

This section summarizes the main contributions of this thesis. Detailed conclusions of each paper can be found at the end of respective paper.

6.1 Regression test selection in the literature

Main contributions of the systematic literature review on regression test selection (paper 1) are:

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

Several important observations were made when analysing the data of the systematic review. Most of the proposed regression test selection techniques are not feasible to scale up to testing of large complex real time systems. Further most of the 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 (e.g. only test suite reduction and not fault detection ability or analysis cost) and the study context is too specific to be easily generalised and applied directly by software developers. Standards for conducting empirical studies, and which measures to evaluate, differ greatly across the studies. Few studies are replicated, and thus the possibility to draw conclusions based on variations in test context is limited.

In order for a practitioner to make use of results of a single study, the study context must be considered and compared to the actual environment into which a technique is supposed to be applied. This is discussed further in (paper 2) which includes proposals for test technique benchmarks e.g.:

1. Define categories for benchmarked methods to avoid comparing "apples with oranges".
2. Define a characterization scheme to capture the relevant degrees of freedom that characterize a test environment.
3. Combine experimental benchmarking results with case studies to analyze both a controlled environment and a real world environment where the interactions between the test technique and its environment can be studied as well.

6.2 Regression testing in practice

In the second part of this work we have focused on industry practice of regression testing which is often based on experience, rather than systematic approaches. Through the means of a survey we wanted to identify challenges and good practices in industry (paper 3). 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. 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.

We have also applied and evaluated a pragmatic strategy, Fix-cache-based regression testing, aimed at improving efficiency of regression testing in a large industrial setting (paper 4). The new method is significantly more efficient, it detects more faults per test case, than the traditional regression test strategy in the current case. The evaluation needs however to be replicated in different settings in order to draw more general conclusions. It should also be evaluated with respect to other aspects of regression test effectiveness such as inclusiveness and precision.

6.3 Software product line testing in the literature

Main contributions of the systematic mapping of research on software product line testing are:

1. A classification scheme for research on software product line testing based on a classification scheme for research on product line variability (Petersen 2008)
2. Overview and classification of research on software product line testing.
3. Overview and quantitative analysis regarding type of research, type of contribution and focus of research.

Most research effort is spent on system testing and the most frequent goal is systematic design of test suites, where generic test cases are derived from a model of the system in domain testing and configured test cases generated from the generic test cases in application testing. Most solutions put requirements on the whole development process which make them hard to introduce in a large organisations. No proposed solution is based on traditional regression test selection techniques. Research contributions are mainly of proposal types and the most frequent types of contributions proposed are methods. There is a lack of empirical evidence supporting the proposals as well as a lack of tools to support the use of them.

7 Conclusions and future work

Even though many strategies are proposed for regression testing as well as for product line testing, there is not much guidance in when to use what strategy and how a strategy should be applied in a specific context. Our previous results (see papers 1, 3 and 5), indicate a need for guidance in selecting strategies suitable in different situations. In this work we have explored and aggregated previous research in order to make results more accessible to practitioners as well as

surveyed and described some important aspects of the real world context aiming at directing future research. The diversity of test situations and its significance to the cost effectiveness of a regression test selection technique are highlighted in (paper 1, 2 and 3). Despite several years of research on regression testing, the gap between research and practice is large and the evidence base is too inconsistent for a meaningful comparison of selection techniques. The mapping between research and the real world context is important and our research will continue to emphasize that.

7.1 Guidance in regression test selection decisions

The main goal is to provide guidance in regression test selection decisions and our continuing work will focus on visualisation of test coverage at different stages of the software development and evaluation of systematic selection strategies in different industrial settings. Decision guidelines should describe the important factors to consider when selecting tests for a particular test situation and show the relations between testing contexts, strategies and its effects on system quality.

7.2 Focus on improvements in product line contexts

The perspective of both regression testing and product line testing will be kept. It is still relevant to compare the product line test selection with regression test selection. The major difference between the two situations is the amount of control. In the case of traditional evolution of software systems, changes may not be well specified and may vary widely in type, size, importance and why and where in the development process they are introduced. The delta between products in the product line, however, is the result of a planned and systematically carried out product strategy. Even though our main focus is on improvements in the product line context, we strive to find solutions applicable in less idealistic product line environments, where traceability links may be broken and variability models not perfectly aligned with the structure of the artefacts throughout the process. Case studies and action research will be conducted along with a company that develops large-scale complex software systems using a product line strategy.

7.3 Visualisation of test coverage

With the current practices as the starting point we want to develop methods for visualizing the test coverage of the different parts of the system and evaluate the relation between the test coverage, different degrees of redundancy, and the quality of those parts of the system. By visualizing the increasing test coverage of the common software base, as an effect of the testing activities at different levels

in both domain and application testing processes, prioritisation and selection of tests are supported. In addition to the achieved test coverage of the common base the delta between the sufficiently tested system and the system under test also need to be visualized. This is a long term goal and solutions need to be iteratively applied and evaluated, thus an action research approach is considered. Sub goals include: evaluate the effects of different granularity of coverage, evaluate the effect of different ways of tracing test cases to the system, evaluate the amount and effects of redundant testing and evaluate the relation between test coverage and the quality of a system.

Granularity and tracing strategy - Most of the suggested coverage measures evaluated in the literature used for regression test selection are code based and not feasible to apply in large scale, real time systems. Test coverage could however refer to coverage at any level such as requirements coverage or variability coverage. An important factor affecting the possibility to identify a proper connection between the test cases and the executed system is how well the structure of the development artefacts is aligned throughout the development process. Selecting an appropriate granularity level of expressing test coverage is a cost benefit trade off which need to be surveyed. A first step is to evaluate the effects of different levels of granularity of coverage in an experimental setting and also evaluate the effects of different ways of tracing test cases to parts of the system in a real life setting.

Quality and redundancy - A measurement of test coverage is useless unless we have some confidence in its correlation with system quality. The relation between software quality and test coverage and the amount and effect of redundant testing is planned to be explored by means of a document study, surveying artefacts from completed industrial product line projects. Variation factors are for example the different ways of defining the system, different levels of granularity and the amount of redundancy.

7.4 Evaluation of pragmatic regression testing strategies

Most of the regression test selection strategies evaluated in literature are based on the assumption that test cases not covering changes in the system are not likely to detect new faults (paper 1). This is of course a logic assumption but the task to keep track of the relation between test cases and changes in the system is not simple in our testing context and it would be interesting to continue evaluating pragmatic solutions not dependent on dataflow analysis or intact traceability links between test cases and code or specifications. In (paper 4) we have previously reported on an empirical evaluation of a pragmatic approach for regression test selection where test cases are connected to the files in the system based on information about fixed faults reported in the error reporting system. We plan for continued evaluation of the fix-cache regression test selection method in terms of

case studies in other companies and in various settings, including variation of parameters defining the method (e.g. the size of the file cache) as well as analysing additional measures such as inclusiveness and precision. Another pragmatic approach is to prioritize test cases based on their execution history (Fazlalizadeh et al. 2009). This strategy is currently being evaluated by means of a post hoc case study similar to the one presented in (paper 4).

8 References

H. Agrawal, J.R. Horgan, E.W. Krauser and S.A. London. Incremental regression testing. In *Proceedings of the Conference on Software Maintenance*. IEEE, 1993, pp. 348–357.

L. Bendix. Fundamental Tasks in Software Development Environments. *Informatica - An International Journal of Computing and Informatics 19,* 3(1995).

J. Bible, G. Rothermel and D.S. Rosenblum. A comparative study of coarse- and finegrained safe regression test-selection techniques. *ACM Transactions on Software Engineering and Methodology 10,* 2(2001) 149–183.

P. Brereton, B.A. Kitchenham, D. Budgen, M. Turner and M. Khalil, Lessons from applying the systematic literature review process within the software engineering domain, *Journal of Systems and Software 80*, 4(2007) 571–583.

L.C. Briand, Y. Labiche and G. Soccar. Automating impact analysis and regression test selection based on UML designs. Technical Report SCE-02-04, Carleton University, <http://www.sce.carleton.ca/Squall>. 2002

I. Burnstein. *Practical Software Testing*, Springer, 2003

Y.-F. Chen, D.S. Rosenblum and K.-P. Vo,. Test tube: a system for selective regression testing. In *Proceedings of the International Conference on Software Engineering*, (Los Alamitos, CA, USA), IEEE, 1994, pp. 211–220.

P. K. Chittimalli and M. J. Harrold. Recomputing coverage information to assist regression testing. *IEEE Transactions on Software Engineering*, *35* 4 (2009), 452-469.

H. Do, S. Elbaum and G. Rothermel. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Software Engineering* 10, 4 (Oct. 2005), 405-435.

S. Easterbrook, J. Singer, M-A. Storey and D. Damian. Selecting empirical methods for software engineering research. Chapter 11 in *Guide to Advanced Empirical Software Engineering,* F. Shull, J. Singer, and D. I. K. Sjøberg, Eds. Springer, 2007, 285-311.

S. Elbaum, A.G. Malishevsky, and G. Rothermel. Test Case Prioritization: A Family of Empirical Studies. *IEEE Transactions on Software Engineering* 28, 2 (Feb. 2002), 159-182.

E. Engström and P. Runeson. A qualitative survey of regression testing practices. In *Proceedings of 11th International Conference on Product Focused Software Development and Process Improvement* (Limerick, Ireland , June 21-23), 2010, in press.

E. Engström and P. Runeson. Software product line testing - a systematic mapping study. *Information and Software Technology*, in press.

E. Engström, P. Runeson and M. Skoglund. 2010a .A systematic review on regression test selection techniques. *Information and Software Technology*, 52, 1(Jan 2010), 14-30.

E. Engström, P. Runeson and G. Wikstrand. 2010b. An empirical evaluation of regression testing based on fix-cache recommendations. In *Proceedings of the hird International Conference on Software Testing, Verification and Validation* (Paris April 7-9). IEEE, 2010, pp. 75-78.

Y. Fazlalizadeh, A. Khalilian, M. Abdollahi Azgomi and S. Parsa. Prioritizing test cases for resource constraint environments using historical test case performance data. In Proceedings of *Computer Science and Information Technology*, 2009, pp. 190-195.

T.L. Graves, M.J. Harrold, J.M. Kim, A. Porter and G. Rothermel. An empirical study of regression test selection techniques. *ACM Transactions on Software Engineering and Methodology* 10, 2 (2001), 184–208

R. Gupta, M.J. Harrold and M.L. Soffa. Program slicing-based regression testing techniques. *Software Testing, Verification and Reliability* 6, 2 (1996), 83–111.

F. Haftmann, D. Kossmann and E. Lo, A framework for efficient regression tests on database applications, *VLDB Journal* 16, 1 (2007), 145–164.

J. Hartmann and D.J. Robson. Techniques for selective revalidation. *IEEE Software* 7, 1, (1990), 31–36.

"IEEE Standard Glossary of Software Engineering Terminology," *IEEE Std 610.12-1990* .

N. Juristo, AM. Moreno, S. Vegas and M. Solari. In search of what we experimentally know about unit testing. *IEEE Software* 23, 6(2006), 72-80.

B.A. Kitchenham. Guidelines for performing Systematic Literature reviews in Software Engineering Version 2.3. Technical Report S.o.C.S.a.M. Software Engineering Group, Keele University and Department of Computer Science University of Durham, 2007.

B.A. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey and S. Linkman. Systematic literature reviews in software engineering - A systematic literature review. *Information and Software Technology*, 51, 1, Special Section - Most Cited Articles in 2002 and Regular Research Papers, (Jan. 2009), 7-15.

B.A. Kitchenham, E. Mendes and G.H. Travassos. Cross versus within-company cost estimation studies: a systematic review. *IEEE Transactions on Software Engineering* 33, 5 (2007) 316–329.

M. Khurum and T. Gorschek. A systematic review of domain analysis solutions for product lines, *Journal of Systems and Software*, 82, 12 (Dec. 2009), 1982-2003.

J. D. McGregor, Testing a Software Product Line. Technical Report, CMU/SEI-2001-TR-022, ESC-TR-2001-022. 2001

A.M. Memon, Using tasks to automate regression testing of GUIs. In Proceeding of *IASTED International Conference on Artificial Intelligence and Applications,* ACTA Press, 2004, pp. 477–482.

E. M. Olimpiew and H. Gomaa. 2005. Model-based testing for applications derived from software product lines. In *Proceedings of the 1st international Workshop on Advances in Model-Based Testing* (St. Louis, Missouri, May 15 - 21). ACM, New York, 2005, pp. 1-7.

A. Orso, S. Nanjuan and M.J. Harrold. Scaling regression testing to large software systems. *Softw. Eng. Notes* , ACM, (2004), 241–251.

K. Petersen, R. Feldt, S. Mujtaba and M. Mattsson . Systematic Mapping Studies in Software Engineering. In Proceedings of *12th International Conference on Evaluation and Assessment in Software Engineering* (Bari, Italy, June 26 – 27), 2008, pp 71–80.

K. Pohl, G. Böckle, and F. van der Linden. *Software Product Line Engineering: Foundations, Principles,and Techniques*, Springer, Heidelberg, August 2005.

S. Reis and A. Metzger and K. Pohl. Integration Testing in Software Product Line Engineering: A Model-Based Technique. In *FASE Lecture Notes in Computer Science 4422*, MB. Dwyer and A. Lopes, Eds. Springer, 2007, 321-335.

G. Rothermel and M.J. Harrold, A safe, efficient regression test selection technique. *ACM Transactions on Software Engineering and Methodology* 6, 2 (1997), 173–210.

G. Rothermel and M.J. Harrold, Analyzing regression test selection techniques, *IEEE Transactions on Software Engineering* 22, 8 (1996), 529–551.

P. Runeson, M. Skoglund and E. Engström. Test Benchmarks -- what is the question?. In Proceedings of *International Conference on Software Testing Verification and Validation Workshop*. IEEE, 2008, pp. 369-371.

M. Skoglund and P. Runeson. A case study of the class firewall regression test selection technique on a large scale distributed software system. In Proceedings of the *International Symposium on Empirical Software Engineering*. 2005, pp. 74-83.

M. Staples and M. Niazi. Experiences using systematic review guidelines. *The Journal of Systems and Software* 80, 9 (2007), 1425–1437.

A. Tevanlinna, J. Taina and R. Kauppinen. Product family testing: a survey. *SIGSOFT Softw. Eng. Notes* 29, 2 (Mar. 2004).

L. White and B. Robinson. Industrial real-time regression testing and analysis using firewalls. In Proceedings of the 20th *IEEE International Conference on Software Maintenance*, 2004, pp. 18–27.

Paper I:

A Systematic Review on Regression Test Selection Techniques

Emelie Engström, Per Runeson and Mats Skoglund

Published in *Journal of Information and Software Technology 52(1):14-30, 2010*

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.

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][6][13][17][20][22][43][49][62][64][67] and specification changes [38][40][54][68], evaluation of selection techniques [48], change impact analysis [44], regression tests for different applications e.g. database applications [18], regression testing of GUIs and test automation [39], and test process enhancement[31]. 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][15][36][65], 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.* [27] evaluate the effects of regression test application frequency, Elbaum *et al.* [11] investigate the impact that different modifications have on regression test selection techniques, several studies examine the ability to reduce regression testing effort [3][11][15][27][36][65][66] and to reveal faults [11][15][27][49].

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][57] and updated accordingly in 2007 [29]. Kitchenham *et al.* recently published a review of 20 systematic reviews in software engineering 2004-2007[28].

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 [8] including regression testing. Their study covered two journals and four conferences 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 [48] and evaluated the, by that time, existing techniques. Juristo *et al.* aggregated results from unit testing experiments [25] 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 [21]. However, as far as we know, no systematic review on regression test selection research has been carried through since the one in 1996 [48]. An early report of this study was published in 2008 [12], 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).[4]

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

[4] In 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.

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?

Answers to these research questions are searched in the published literature using the procedures of systematic literature reviews as proposed by Kitchenham [29].

2.2 Sources of information

In order to gain a broad perspective, as recommended in Kitchenham's guidelines [29], 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 [7]. 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.* [8]. 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 [29]. However, only papers written in English are included. The initial search located 2 923 potentially relevant papers.

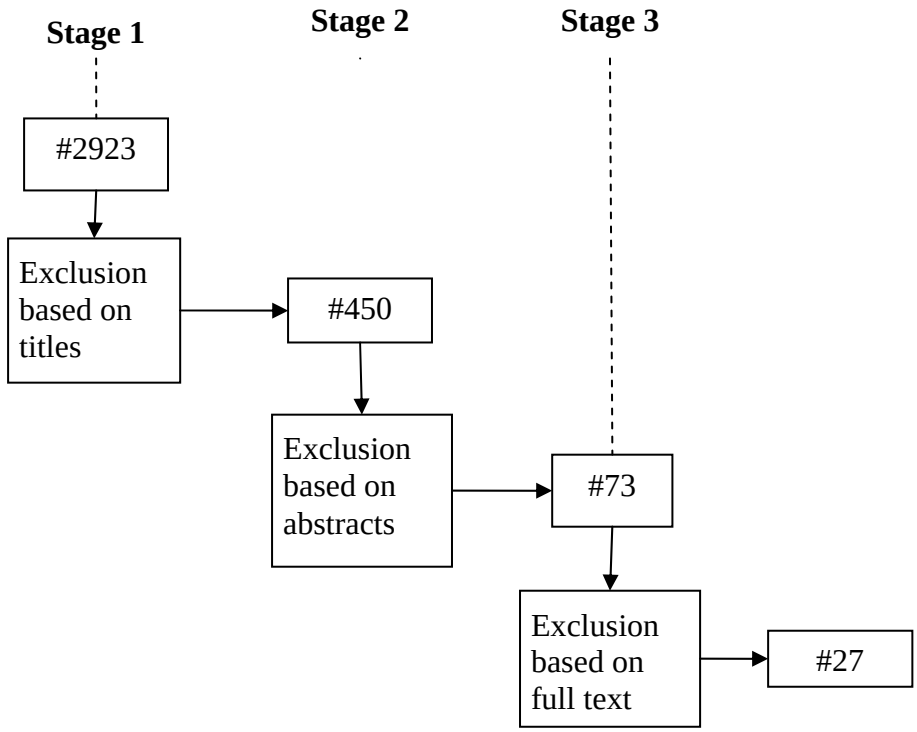

Figure 1. Study selection procedure

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.

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.* [8]. 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 [33] 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 3.1. 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 [55].
- 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 [69].

Surveys and literature reviews were also considered in the systematic review, e.g. [48] and [25], 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 3.4.

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.

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

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 made by two researchers in parallel and uncertainties were resolved in discussions. Results are presented in Figures 5 – 8 in Section 3.5.

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"	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 [8]. 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 were 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 8 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 REF1	Own unspecified	Exp	S
S2	P2	Bible et al. (2001) [3]	T7, T8 REF1	*7x Siemens,* Small constructed programs, constructed, realistic non-coverage based test suites	Exp	S
S3	P2	Bible et al. (2001)[3]	T7, T8 REF1	*Space,* Real application, real faults, constructed test cases	Exp	S
S4	P2	Bible et al. (2001) [3]	T7, T8 REF1	*Player*, One module of a large software system constructed realistic test suites	Exp	M
S5	P3	Elbaum et al. (2003)	T2, T4, T18 REF1	*Bash, Grep, Flex* and *Gzip*, Real, non-trivial C	CS (Mult)	M

Study ID	Publica-tion ID	Reference	Techniques	Artifacts	Type of study	Size of study
		[11]		program, constructed test suites		
S6	P4	Frankl et al. (2003) [14]	T7, T10 REF1	*7xSiemens,* Small constructed programs, constructed, realistic, non-coverage based test suites	Exp	S
S7	P5	Graves et al. (2001) [15]	T1, T2, T7 REF1, REF2, REF3, REF4	*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) [19]	T15 REF1	*Siena, Jedit, JMeter, RegExp,* Real programs, constructed faults	Exp	S
S9	P7	Kim et al. (2005)[27]	T2, T7, T8 REF1, REF2, REF3, REF4	*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) [32]	T15 REF1	*Classes in .net framework,* Open source, real test cases	Exp	S
S11	P9	Mansour et al. (2001) [36]	T4, T5, T6, T12	20 small sized Modules	Exp	S
S12	P10	Mao and Lu (2005) [38]	T16, T17, T24 REF1	*Triangle, eBookShop, ShipDemo,* Small Constructed programs	CS	S
S13	P11	Orso et al. (2004) [41]	T9, T15, T19 REF1	*Jaba, Daikon, JBoss,* Real-life programs, original test suites	Exp	M L
S14	P12	Pasala and Bhowmick (2005) [42]	T20 REF1	*Internet Explorer* (client), *IIS* (web server), *application* (app. Server), An existing browser based system, real test cases	CS	NR
S15	P13	Rothermel	T7	*7xSiemens*, Small	Exp	S

Study ID	Publica-tion ID	Reference	Techniques	Artifacts	Type of study	Size of study
		and Harrold (1997) [49]	REF1	constructed programs, constructed, realistic non-coverage based test suites		
S16	P13	Rothermel and Harrold (1997) [49]	T7 REF1	*Player*, One module of a large software system constructed realistic test suites	Exp	M
S17	P14	Rothermel and Harrold (1998) [50]	T7 REF1	*7xSiemens,* Small constructed programs, constructed, realistic non-coverage based test suites	Exp	S
S18	P14	Rothermel and Harrold (1998) [50]	T7 REF1	*7xSiemens,* Small constructed programs, constructed, realistic non-coverage based test suites	Exp	S
S19	P14	Rothermel and Harrold (1998) [50]	T7 REF1	*7xSiemens*, Small constructed programs, constructed, realistic non-coverage based test suites;	Exp	S
S20	P14	Rothermel and Harrold (1998) [50]	T7 REF1	*Player,* One module of a large software system constructed realistic test suites	Exp	M
S21	P14	Rothermel and Harrold (1998) [50]	T7 REF1	*Commerercial,* Real application, real test suite	Exp	S
S22	P15	Rothermel et al. (2002)[45]	T8, T18 REF1	*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)[46]	T2, T8, T18 REF1	*Bash,* Open-source, real and constructed test cases	Exp	M
S24	P16	Rothermel et al. (2004) [46]	T2, T8, T18 REF1	*Emp-server,* Open-source, server component, constructed test cases	Exp	M
S25	P17	Skoglund and Runeson (2005) [56]	T9, T21 REF1	*Swedbank,* Real, large scale, distributed, component-based, J2EE system, constructed, scenario-based test cases	CS	L

Study ID	Publica-tion ID	Reference	Techniques	Artifacts	Type of study	Size of study
S26	P18	Vokolos and Frankl (1998) [65]	T10 REF1	*ORACOLO2,* Real industrial subsystems, real modifications, constructed test cases	CS	M
S27	P19	White and Robinson (2004) [61]	T3 REF5	*14 real ABB projects,* Industrial, Real-time system	CS	L
S28	P19	White and Robinson (2004) [61]	T9 REF5	*2 real ABB projects*, Industrial, Real-time system	CS	L
S29	P20	White et al. (2005) [60]	T3, T9, T25	OO-telecommunication software system	CS	S
S30	P20	White et al. (2005) [60]	T3, T9, T25	OO – real-time software system	CS	L
S31	P21	Willmor and Embury (2005)[63]	T7, T22, T23 REF1	*Compiere, James, Mp3cd browser,* Open source systems, real modifications	CS	NR
S32	P22	Wong et al. (1997)[66]	T13 REF1	*Space,* Real application, real faults, constructed test cases	CS	S
S33	P23	Wu et al. (1999) [67]	T14 REF1	*ATM-simulator,* small constructed program	CS	S
S34	P23	Wu et al. (1999) [67]	T14 REF1	Subsystem of a fully networked supervisory control and data analysis system	CS	M
S35	P24, P25, P26	Zheng et al. (2005) [71], Zheng et al. (2006) [72] Zheng (2005) [70]	T26, T28 REF1	*ABB-internal,* Real C/C++ application	CS	M
S36	P27, P25	Zheng et al. (2006) [74], Zheng et al. (2006) [72]	T27, T28 REF1	*ABB-internal,* Real C/C++ application	CS	M

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) [46] 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 2.5, 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 3.3 as sources of information about the techniques as such (Table 8).

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**

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. 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.

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

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

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.* [8], but apparently not on regression test selection during the studied time period.

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**

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 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).

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		

The regression test selection techniques have been published from 1988 to 2006, as shown in Fig 2 and Table 8. 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).

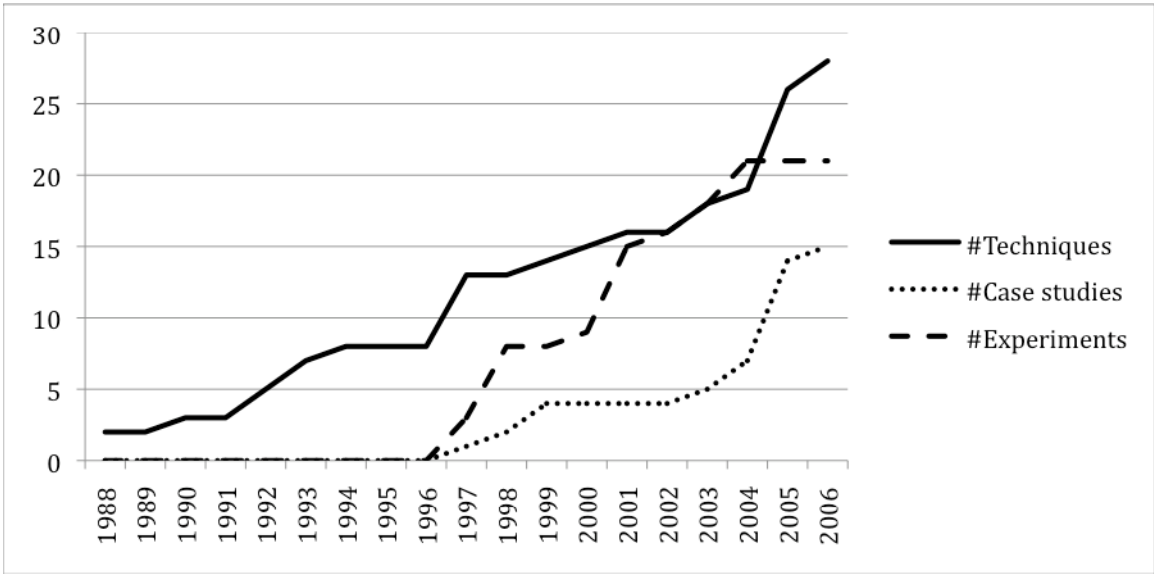

Fig 2. Accumulated number of published techniques, case studies and experiments.

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 [8], 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)

3.3.1 Overview

Table 8 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 2.5.

Table 8. Techniques for regression test selection

Technique	Origin	Description	Evaluated in study
T1	Harrold and Soffa (1988) [20]	Dataflow-coverage-based	S7
T2	Fischer et al. (1981) [13] Hartman and Robson (1988) [22]	Modification-focused, minimization, branch and bound algorithm	S5, S7, S9, S23, S24
T3	Leung and White (1990) [35]	Procedural-design firewall	S27, S29, S30
T4	Gupta et al. (1992) [16]	Coverage-focused, slicing	S1, S5, S11
T5	White and Leung (1992) [62]	Firewall	S1, S11
T6	Agraval et al. (1993)[1]	Incremental	S1, S11

Technique	Origin	Description	Evaluated in study
T7	Rothermel and Harrold (1993)[47]	Viewing statements, DejaVu	S2 -S4, S6, S7, S9, S15 – S21, S31
T8	Chen and Rosenblum (1994) [6]	Modified entity - TestTube	S2 - S4, S9, S22 - 24
T9	Pei et al. (1997) [43] White and Abdullah (1997) [59]	High level – identifies changes at the class and interface level	S13, S25, S28 - S30
T10	Vokolos and Frankl (1997) [64]	Textual Differing - Pythia	S6, S26
T11	Mansour and Fakih (1997) [37]	Genetic algorithm	S1
T12	Mansour and Fakih (1997) [37]	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) [19] Koju et al. (2003) [32]	Edge level - identifies changes at the edge level	S8, S10, S13
T16	Orso et al. (2001) [40]	Use of metadata to represent links between changes and Test Cases	S12
T17	Sajeev et al. (2003) [54]	Use of UML (OCL) to describe information changes	S12
T18	Elbaum et al. (2003) [11]	Modified-non-core Same as T8 but ignoring core functions	S5, S22
T19	Orso et al. (2004) [41]	Partitioning and selection Two Phases	S13
T20	Pasala and Bhowmick (2005) [42]	Runtime dependencies captured and modeled into a graph (CIG)	S14
T21	Skoglund and Runeson (2005) [56]	Change based selection	S25
T22	Willmor and Embury (2005)[63]	Test selection for DB-driven applications (extension of T7) combined safety	S31
T23	Willmor and Embury (2005) [63]	Database safety	S31
T24	Mao and Lu (2005) [38]	Enhanced representation of change information	S12
T25	White et al. (2005) [60]	Extended firewall additional data-paths	S29, S30

Technique	Origin	Description	Evaluated in study
T26	Zheng (2005)[71]	I-BACCI v.1	S35
T27	Zheng et al. (2006)[74]	I-BACCI v.2 (firewall + BACCI)	S36
T28	Zheng et al. (2006) [74]	I-BACCI v.3	S35, S36
REF1	Leung and White (1989) [34]	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

3.3.2 Development history

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

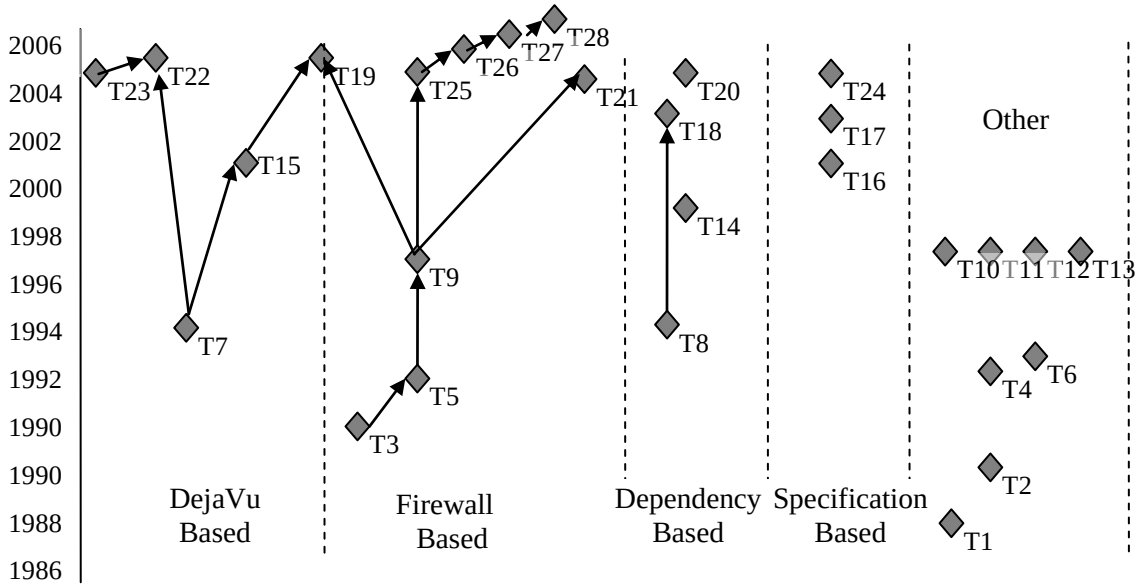

Fig 3. Evolution of techniques

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 [35]. An empirical evaluation used a changed version (T5). The technique was adapted to object-oriented languages T9 in two similar ways [43][59] 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 [47] (T7), sometimes referred to as *DejaVu*. This technique has later been adopted to object-oriented languages T15 (for C++ [51], and for Java[19][32]) and also further extended for MSIL code [32]. 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 [63].

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

3.3.3 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 [11] 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 [11] 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 [19][41], component based software, T17, T20, T24 and T28 [38][42][72][74], or database-driven applications, T22, [63]. 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 [19]. In [58] 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 [19], nor the adapted version, used in [32]. 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. Rothermel and Harrold present a framework for analyzing regression test selection techniques [48], including evaluation criteria for the techniques: inclusiveness, precision, efficiency and generality. Graves et al. [15] present a classification scheme where techniques are classified as Minimization, Safe, Dataflow-Coverage-based, Ad-hoc/Random or Retest-All techniques. Orso et al. [41] 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 9.

Table 9. Overview of properties for each technique.

	Applicability		Method			Properties	
Tech-nique	**Type of Lan-guage**	**Type of Soft-ware**	**Input**	**Approach**	**Granularity**	**Detect ion Ability**	**Cost Reduc-tion**
	Proc= Procedural language Ind = Independent OO = Object oriented	Comp = Component based DB = Database driven	SC = Source code IM = Intermediate code for virtual machines BIN = Machine code Spec = Input of a certain format	CF = Control flow FW = Fire wall Slicing Dep = Dependency based Genetic SimAn= Simulated annealing	Stm = statement Func = Function Class Module Component	Safe	Min = Minimization
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	

	Applicability		Method			Properties	
Tech-nique	**Type of Lan-guage**	**Type of Soft-ware**	**Input**	**Approach**	**Granularity**	**Detect ion Ability**	**Cost Reduc-tion**
	Proc= Procedural language Ind = Independent OO = Object oriented	Comp = Component based DB = Database driven	SC = Source code IM = Intermediate code for virtual machines BIN = Machine code Spec = Input of a certain format	CF = Control flow FW = Fire wall Slicing Dep = Dependency based Genetic SimAn= Simulated annealing	Stm = statement Func = Function Class Module Component	Safe	Min = Minimization
T23	OO	DB	SC	CF	Stm	Safe[5]	
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		

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).

3.5.1 Types of empirical evidence

Table 10 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 [23], often referred to as the Siemens programs, which are made available through the software infrastructure repository[6] presented by Do *et al.* [8].

[5] Safe only in DB-state

[6] http://sir.unl.edu

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.

Table 10. 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**

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 2.4 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 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.

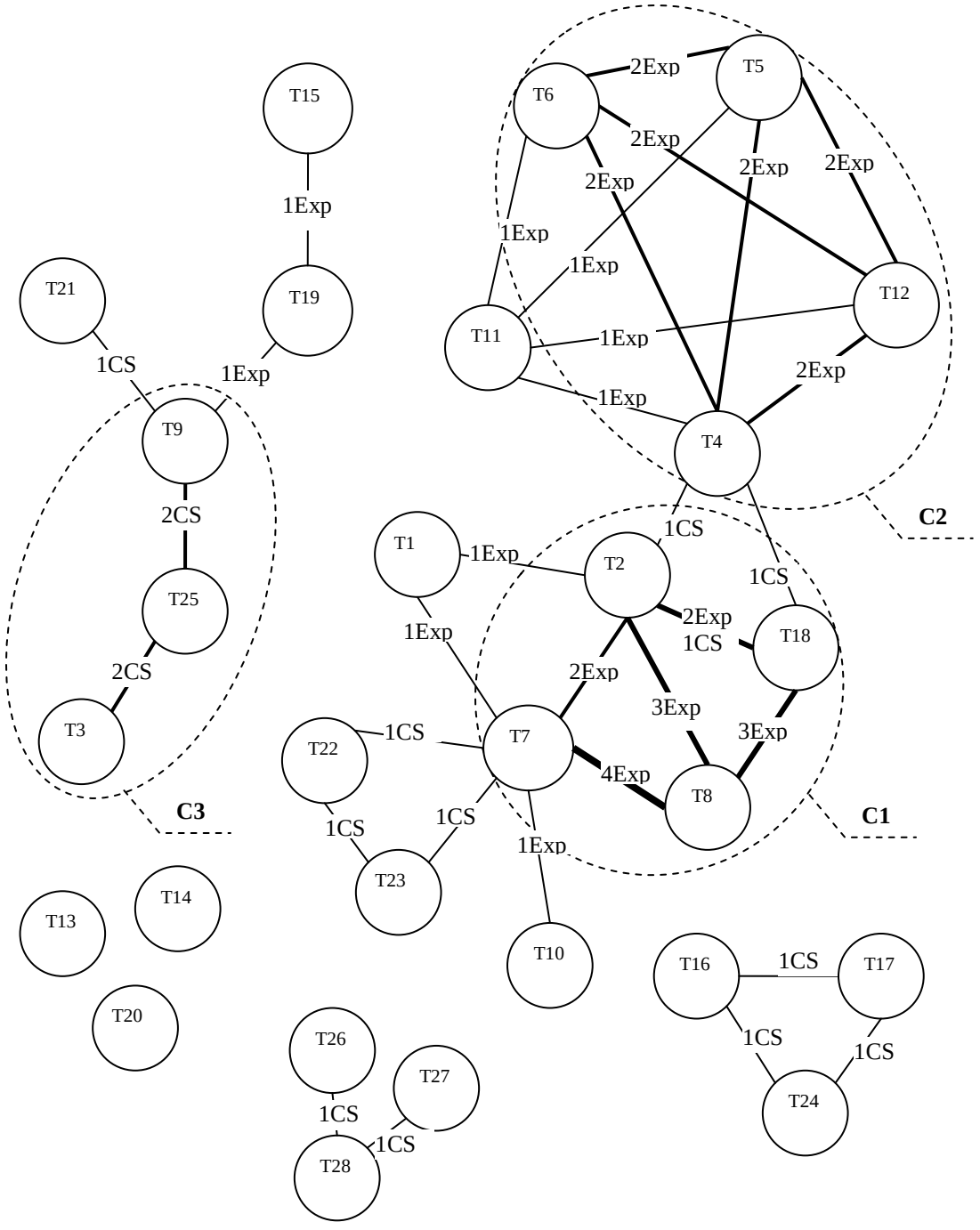

Fig 4. Techniques related to each other through empirical comparisons

Three clusters of techniques have been evaluated sufficiently to allow for meaningful comparison, see Fig 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 2.6, and with respect to evaluation criteria, as desribed below.

3.5.2 Evaluation criteria

Do and Rothermel proposed a cost model for regression testing evaluation [9]. 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 11 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.

Table 11. Use of evaluation metrics in the studies

	Evaluated Metrics	Number	%	Rothermel framework [48]
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	

Several of the studies concerning reduction of number of test cases are only compared to retest all (S8, S10, S14-S21, S26, S32-S34) [19], [32], [42], [49], [50], [65], [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 [26]. 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 and cost reduction. Rothermel proposed a framework for evaluation of regression test selection techniques [48] 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 11. 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.

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.

3.6.1 Cost reduction

Fig 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 however, 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 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 Fig 6.

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

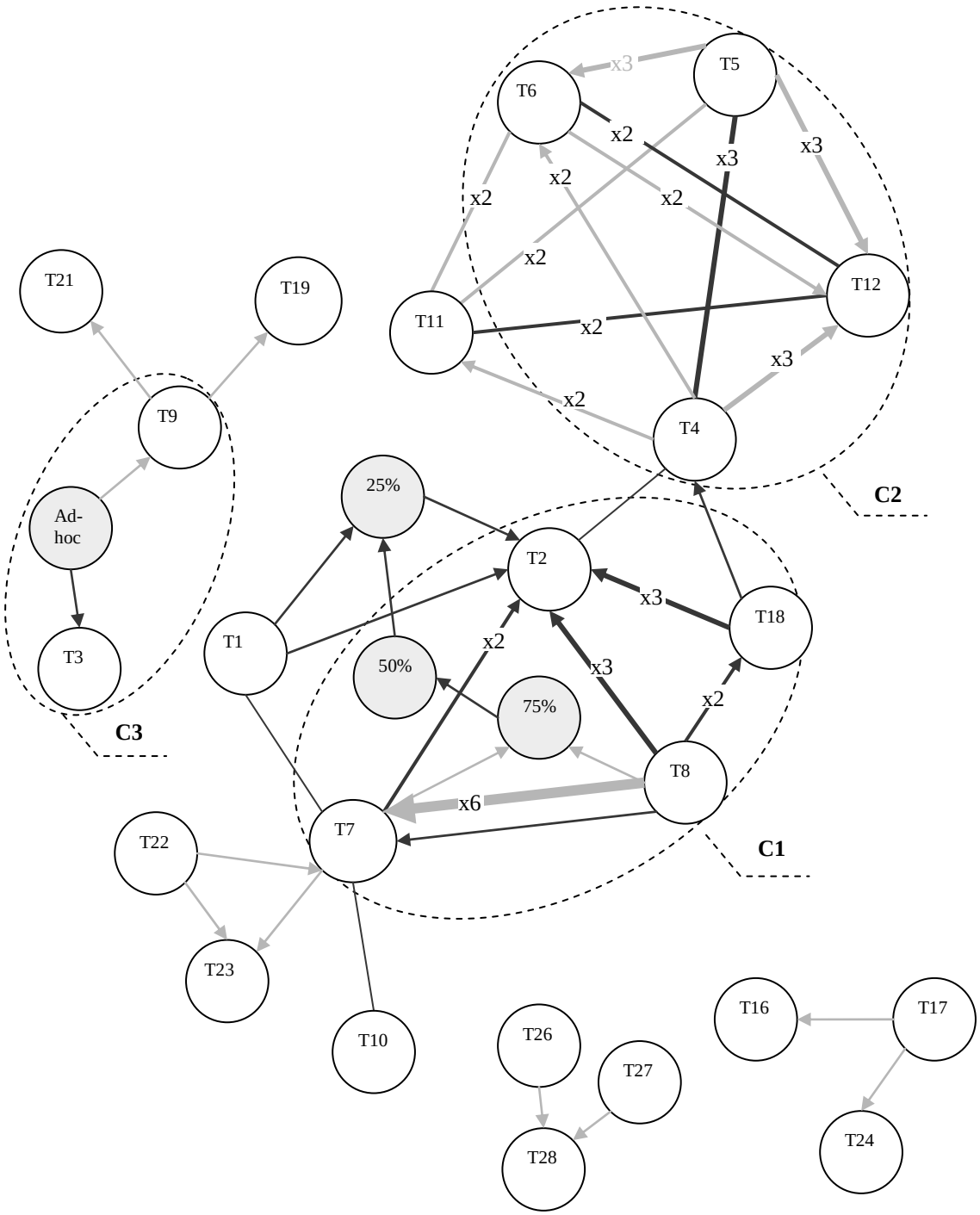

Fig 5. Empirical results for Cost Reduction, including Test Execution Time, Test Suite Reduction and Precision.

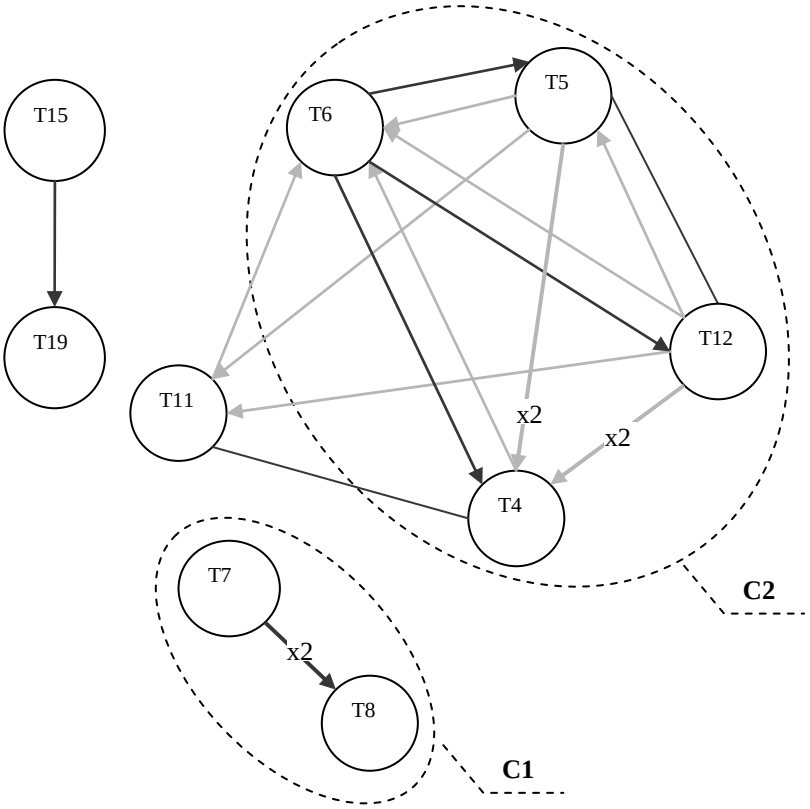

Fig 6. Empirical results for Test Selection Time

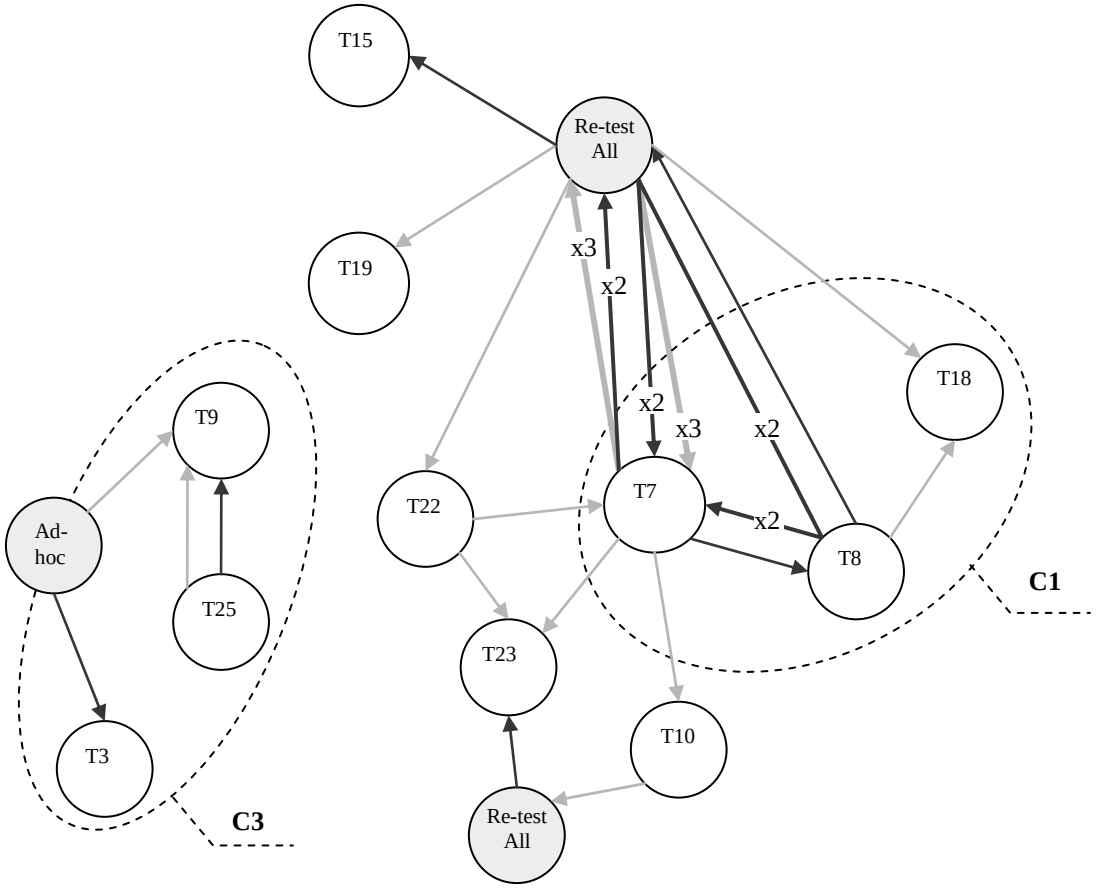

Fig 7. Empirical results for Total Time

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 Fig 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.

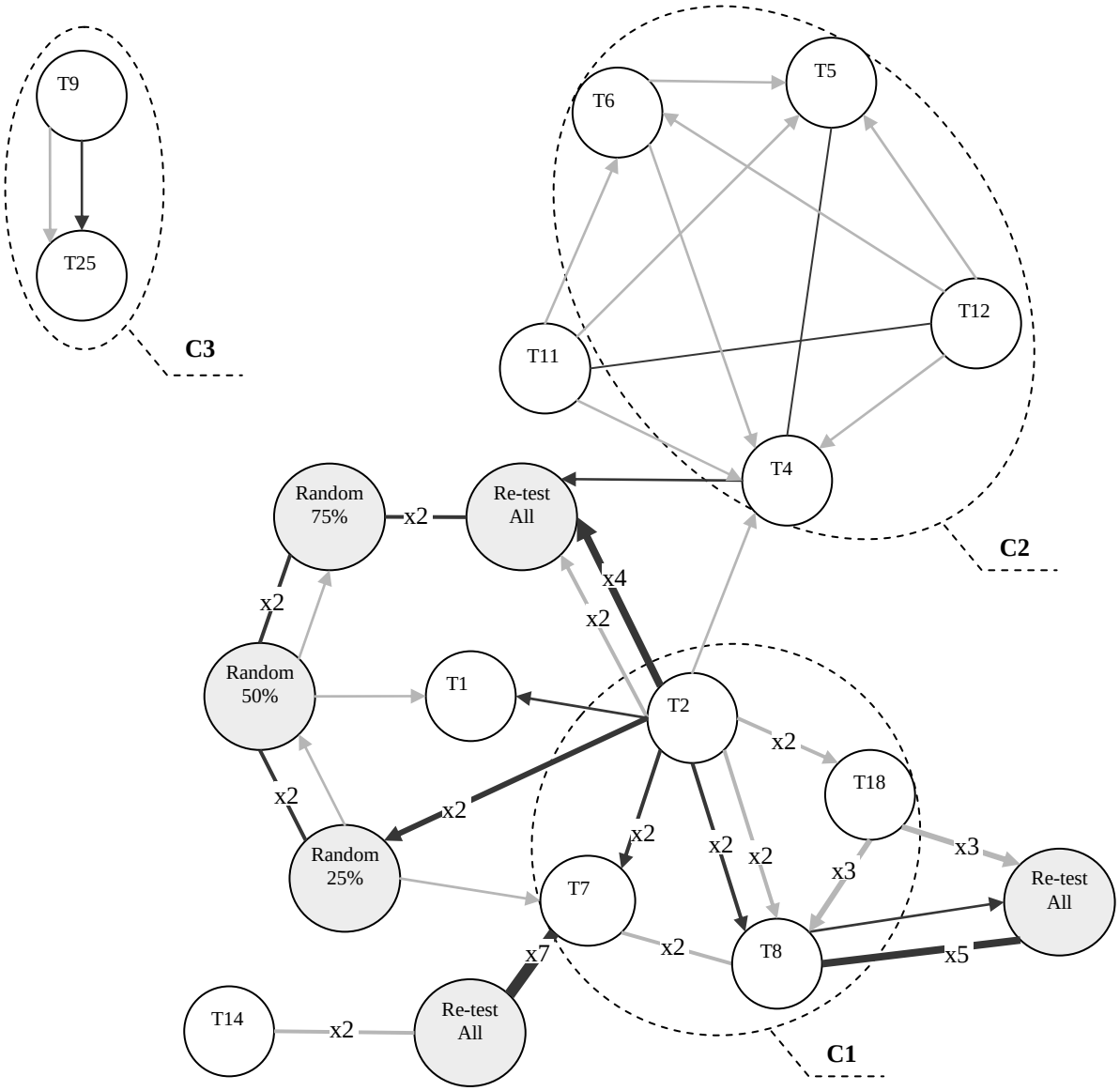

Fig 8. Empirical results for Fault Detection Effectiveness

3.6.2 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 test case-related as well as fault-related detection effectiveness are presented in Fig 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 3.4). 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 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 3.4) 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 [48], 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 9. *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 [53]. *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 [50], the level of granularity of tests [3], as well as testing frequency [27] and the effect of changes [11]. 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 [8] 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 [24]for experiments and Runeson and Höst [52] 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.

6 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.

7 References

[1] Agrawal, H., Horgan, J.R., Krauser, E.W., and London, S.A. 1993. Incremental regression testing. In Proceedings. Conference on Software Maintenance 1993. CSM-93 (Cat. No.93CH3360-5). IEEE Comput. Soc. Press, 348-57.

[2] Baradhi, G. and Mansour, N. 1997. A comparative study of five regression testing algorithms. Software Engineering Conference, 1997. Proceedings. 1997 Australian. 174-182.

[3] Bible, J., Rothermel, G., and Rosenblum, D.S. 2001. A comparative study of coarse- and fine-grained safe regression test-selection techniques. ACM Transactions on Software Engineering and Methodology. 10(2), 149-183.

[4] Binkley, D. 1998. The application of program slicing to regression testing. Information and Software Technology. 40(11-12), 583-94.

[5] Brereton, P., Kitchenham, B.A., Budgen, D., Turner, M., and Khalil, M. 2007. Lessons from applying the systematic literature review process within the software engineering domain. Journal of Systems and Software. 80(4), 571-83.

[6] Chen, Y.-F., Rosenblum, D.S., and Vo, K.-P. 1994. Test tube: A system for selective regression testing. In Proceedings - International Conference on Software Engineering. IEEE, Los Alamitos, CA, USA, 211-220.

[7] Dieste, O., Grimán, A., and Juristo, N. 2008. Developing search strategies for detecting relevant experiments. Empirical Software Engineering.

[8] Do, H. Elbaum, S. and Rothermel, G. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact, Empirical Software Engineering, An International Journal, V. 10, No. 4, October 2005

[9] Do, H. and Rothermel, G., An empirical study of regression testing techniques incorporating context and lifecycle factors and improved cost-benefit models, Proceedings of the ACM SIGSOFT Symposium on Foundations of Software Engineering, November 2006, pages 141-151.

[10] Dybå, T., Dingsöyr, T., and Hanssen, G.K. 2007. Applying Systematic Reviews to Diverse Study Types: An Experience Report. In First International Symposium on Empirical Software Engineering and Measurement, 2007, ESEM 2007. 225-234.

[11] Elbaum, S., Kallakuri, P., Malishevsky, A., Rothermel, G., and Kanduri, S. 2003. Understanding the effects of changes on the cost-effectiveness of regression testing techniques. Software Testing, Verification and Reliability. 13(2), 65-83.

[12] Engström, E., Skoglund, Mats, Runeson, Per. 2008. Empirical Evaluations of Regression Test Selection Techniques: A Systematic Review. ESEM 08

[13] Fischer, K., Raji, F., and Chruscicki, A. 1981. A methodology for retesting modified software. In NTC '81. IEEE 1981 National Telecommunications Conference. Innovative Telecommunications - Key to the Future. IEEE, 6-3.

[14] Frankl, P.G., Rothermel, G., Sayre, K., and Vokolos, F.I. 2003. An empirical comparison of two safe regression test selection techniques. Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on. 195-204.

[15] Graves, T.L., Harrold, M.J., Kim, J.M., Porter, A., and Rothermel, G. 2001. An empirical study of regression test selection techniques. ACM Transactions on Software Engineering and Methodology. 10(2), 184-208.

[16] Gupta, R., Harrold, M.J., and Soffa, M.L. 1992. An approach to regression testing using slicing. In Conference on Software Maintenance 1992 (Cat.No.92CH3206-0). IEEE Comput. Soc. Press, 299-308.

[17] Gupta, R., Harrold, M.J., and Soffa, M.L. 1996. Program slicing-based regression testing techniques. Software Testing, Verification and Reliability. 6(2), 83-111.

[18] Haftmann, F., Kossmann, D., and Lo, E. 2007. A framework for efficient regression tests on database applications. VLDB Journal. 16(1), 145-64.

[19] Harrold, M.J., Jones, J.A., Tongyu, L., Donglin, L., Orso, A., Pennings, M., Sinha, S., Spoon, S.A., and Gujarathi, A. 2001. Regression test selection for Java software. In SIGPLAN Not. (USA). ACM, 312-26.

[20] Harrold, M.J. and Souffa, M.L. 1988. An incremental approach to unit testing during maintenance. In Proceedings of the Conference on Software Maintenance - 1988 (IEEE Cat. No.88CH2615-3). IEEE Comput. Soc. Press, 362-7.

[21] Hartmann, J. and Robson, D.J. 1988. Approaches to regression testing. In Proceedings of the Conference on Software Maintenance - 1988 (IEEE Cat. No.88CH2615-3). IEEE Comput. Soc. Press, 368-72.

[22] Hartmann, J. and Robson, D.J. 1990. Techniques for selective revalidation. IEEE Software. 7(1), 31-6.

[23] Hutchins, M., Foster, H., Goradia, T., and Ostrand, T. 1994. Experiments on the effectiveness of dataflow- and control-flow-based test adequacy criteria. In ICSE-16. 16th International Conference on Software Engineering (Cat. No.94CH3409-0). IEEE Comput. Soc. Press, 191-200.

[24] Jedlitschka, A.and Pfahl, D. 2005. Reporting Guidelines for Controlled Experiments in Software Engineering, In Proceedings of ACM/ IEEE International Symposium on Empirical Software Engineering, pp 95-104

[25] Juristo, N., Moreno, A.M., Vegas, S., and Solari, M. 2006. In search of what we experimentally know about unit testing [software testing]. IEEE Software. 23(6), 72-80.

[26] Kampenes Vigdis, B., Dybå, T., Hannay Jo, E., and Sjöberg Dag, I.K. 2007. A systematic review of effect size in software engineering experiments. Information and Software Technology. 49(11-12), 1073-1073.

[27] Kim, J.-M., Porter, A., and Rothermel, G. 2005. An empirical study of regression test application frequency. Software Testing, Verification and Reliability. 15(4), 257-279.

[28] Kitchenham, B., Brereton, O.P., Budgen, D., Turner, M., Bailey, J., and Linkman, S. 2009. Systematic literature reviews in software engineering - A systematic literature review. Information and Software Technology. Volume 51(Issue 1), Pages 7-15.

[29] Kitchenham, B.A. 2007. Guidelines for performing Systematic Literature reviews in Software Engineering Version 2.3. Technical Report S.o.C.S.a.M. Software Engineering Group, Keele University and Department of Computer Science University of Durham.

[30] Kitchenham, B.A., Mendes, E., and Travassos, G.H. 2007. Cross versus within-company cost estimation studies: a systematic review. IEEE Transactions on Software Engineering. 33(5), 316-29.

[31] Klosch, R.R., Glaser, P.W., and Truschnegg, R.J. 2002. A testing approach for large system portfolios in industrial environments. Journal of Systems and Software. 62(1), 11-20.

[32] Koju, T., Takada, S., and Doi, N. 2003. Regression Test Selection based on Intermediate Code for Virtual Machines. In Conference on Software Maintenance. Institute of Electrical and Electronics Engineers Inc., 420-429.

[33] Landis, J.R. and Gary, G.K. 1977. The Measurement of Observer Agreement for Categorical Data. Biometrics. 33(1), 159-174.

[34] Leung, H.K.N. and White, L. 1990. Insights into testing and regression testing global variables. Journal of Software Maintenance: Research and Practice. 2(4), 209-22.

[35] Leung, H.K.N. and White, L. 1990. A study of integration testing and software regression at the integration level. In Proceedings. Conference on Software Maintenance 1990 (Cat. No.90CH2921-5). IEEE Comput. Soc. Press, 290-301.

[36] Mansour, N., Bahsoon, R., and Baradhi, G. 2001. Empirical comparison of regression test selection algorithms. The Journal of Systems and Software. 57(1), 79-90.

[37] Mansour, N. and El-Fakih, K. 1997. Natural optimization algorithms for optimal regression testing. In Proceedings - IEEE Computer Society's International Computer Software & Applications Conference. IEEE, Los Alamitos, CA, USA, 511-514.

[38] Mao, C. and Lu, Y. 2005. Regression testing for component-based software systems by enhancing change information. In Proceedings. 12th Asia-Pacific Software Engineering Conference. IEEE Computer Society, 8 pp.

[39] Memon, A.M. 2004. Using tasks to automate regression testing of GUIs. In IASTED International Conference on Artificial Intelligence and Applications - AIA 2004. ACTA Press, 477-82.

[40] Orso, A., Harrold, M.J., Rosenblum, D., Rothermel, G., Soffa, M.L., and Do, H. 2001. Using component metacontent to support the regression testing of component-based software. In Proceedings IEEE International Conference on Software Maintenance. ICSM 2001. IEEE Comput. Soc, 716-25.

[41] Orso, A., Nanjuan, S., and Harrold, M.J. 2004. Scaling regression testing to large software systems. In Softw. Eng. Notes (USA). ACM, 241-51.

[42] Pasala, A. and Bhowmick, A. 2005. An approach for test suite selection to validate applications on deployment of COTS upgrades. In Proceedings - Asia-Pacific Software Engineering Conference, APSEC. IEEE Computer Society, Los Alamitos, CA 90720-1314, United States, 401-407.

[43] Pei, H., Xiaolin, L., Kung, D.C., Chih-Tung, H., Liang, L., Toyoshima, Y., and Chen, C. 1997. A technique for the selective revalidation of OO software. Journal of Software Maintenance: Research and Practice. 9(4), 217-33.

[44] Ren, X., Shah, F., Tip, F., Ryder, B.G., and Chesley, O. 2004. Chianti: A tool for change impact analysis of java programs. In 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA'04. Association for Computing Machinery, New York, NY 10036-5701, United States, 432-448.

[45] Rothermel, G., Elbaum, S., Malishevsky, A., Kallakuri, P., and Davia, B. 2002. The impact of test suite granularity on the cost-effectiveness of regression testing. In Proceedings - International Conference on Software Engineering. Institute of Electrical and Electronics Engineers Computer Society, 130-140.

[46] Rothermel, G., Elbaum, S., Malishevsky, A.G., Kallakuri, P., and Xuemei, Q. 2004. On test suite composition and cost-effective regression testing. ACM Transactions on Software Engineering and Methodology. 13(3), 227-331.

[47] Rothermel, G. and Harrold, M.J. 1993. A safe, efficient algorithm for regression test selection. Software Maintenance ,1993. CSM-93, Proceedings., Conference on. 358-367.

[48] Rothermel, G. and Harrold, M.J. 1996. Analyzing regression test selection techniques. IEEE Transactions on Software Engineering. 22(8), 529-51.

[49] Rothermel, G. and Harrold, M.J. 1997. A safe, efficient regression test selection technique. ACM Transactions on Software Engineering and Methodology. 6(2), 173-210.

[50] Rothermel, G. and Harrold, M.J. 1998. Empirical studies of a safe regression test selection technique. IEEE Transactions on Software Engineering. 24(6), 401-19.

[51] Rothermel, G., Harrold, M.J., and Dedhia, J. 2000. Regression test selection for C++ software. Journal of Software Testing Verification and Reliability. 10(2), 77-109.

[52] Runeson, P. and Höst, M. Guidelines for conducting and reporting case study research in software engineering, Empirical Software Engineering, 14(2):131-164, 2009.

[53] Runeson, P., Skoglund, M. and Engström, E. Test Benchmarks – what is the question?, TestBench Workshop at International Conference on Software Testing, Verification and Validation, Lillehammer, Norway, April 2008.

[54] Sajeev, A.S.M. and Wibowo, B. 2003. Regression test selection based on version changes of components. In Tenth Asia-Pacific Software Engineering Conference. IEEE Comput. Soc, 78-85.

[55] Shadish, T., Cook, T., and Campbell, D. 2002. Experimental and Quasi-Experimental Designs - for Generalized Causal Inference. 2 ed, Boston: Houghton Mifflin Company. 623.

[56] Skoglund, M. and Runeson, P. 2005. A case study of the class firewall regression test selection technique on a large scale distributed software system. In 2005 International Symposium on Empirical Software Engineering (IEEE Cat. No. 05EX1213). IEEE, 10 pp.

[57] Staples, M. and Niazi, M. 2007. Experiences using systematic review guidelines. The Journal of Systems & Software. 80(9), 1425-37.

[58] Toshihiko, K., Shingo, T., and Norihisa, D. 2003. Regression test selection based on intermediate code for virtual machines. In Proceedings International Conference on Software Maintenance ICSM 2003. IEEE Comput. Soc, 420-9.

[59] White, L. and Abdullah, K. 1997. A firewall approach for the regression testing of object-oriented software. Software Quality Week

[60] White, L., Jaber, K., and Robinson, B. 2005. Utilization of extended firewall for object-oriented regression testing. In IEEE International Conference on Software Maintenance, ICSM. IEEE Computer Society, Los Alamitos, CA 90720-1314, United States, 695-698.

[61] White, L. and Robinson, B. 2004. Industrial real-time regression testing and analysis using firewalls. Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on. 18-27.

[62] White, L.J. and Leung, H.K.N. 1992. A firewall concept for both control-flow and data-flow in regression integration testing. In Conference on Software Maintenance 1992 (Cat.No.92CH3206-0). IEEE Comput. Soc. Press, 262-71.

[63] Willmor, D. and Embury, S.M. 2005. A safe regression test selection technique for database-driven applications. In Proceedings of the 21st IEEE International Conference on Software Maintenance. IEEE Comput. Soc, 421-30.

[64] Vokolos, F.I. and Frankl, P.G. 1997. Pythia: a regression test selection tool based on textual differencing. In Reliability, Quality and Safety of Software-Intensive Systems. IFIP TC5 WG5.4 3rd International Conference. Chapman & Hall, 3-21.

[65] Vokolos, F.I. and Frankl, P.G. 1998. Empirical evaluation of the textual differencing regression testing technique. In Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272). IEEE Comput. Soc, 44-53.

[66] Wong, W.E., Horgan, J.R., London, S., and Agrawal, H. 1997. A study of effective regression testing in practice. In Proceedings. The Eighth International Symposium on Software Reliability Engineering (Cat. No.97TB100170). IEEE Comput. Soc, 264-74.

[67] Wu, Y., Chen, M.-H., and Kao, H.M. 1999. Regression testing on object-oriented programs. In Proceedings 10th International Symposium on Software Reliability Engineering (Cat. No.PR00443). IEEE Comput. Soc, 270-9.

[68] Yanping, C., Robert, L.P., and Sims, D.P. 2002. Specification-based regression test selection with risk analysis. Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research. IBM Press.

[69] Yin, R.K. 2003. Case Study Research - Design and Methods Applied Social Research Methods Series, ed. D.J.R. Leonard Bickman. Vol. 5, London: Sage Publications.

[70] Zheng, J. 2005. In regression testing selection when source code is not available. Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering. ACM.

[71] Zheng, J., Robinson, B., Williams, L., and Smiley, K. 2005. An initial study of a lightweight process for change identification and regression test selection when source code is not available. In Proceedings - International Symposium on Software Reliability Engineering, ISSRE. IEEE Computer Society, 225-234.

[72] Zheng, J., Robinson, B., Williams, L., and Smiley, K. 2006. Applying regression test selection for COTS-based applications. In Proceedings - International Conference on Software Engineering. Institute of Electrical and Electronics Engineers Computer Society, Piscataway, NJ 08855-1331, United States, 512-521.

[73] Zheng, J., Robinson, B., Williams, L., and Smiley, K. 2006. An initial study of a lightweight process for change identification and regression test selection when source code is not available. In Proceedings. 16th IEEE International Symposium on Software Reliability Engineering. IEEE Computer Society, 10 pp.

[74] Zheng, J., Robinson, B., Williams, L., and Smiley, K. 2006. A lightweight process for change identification and regression test selection in using COTS components. In Proceedings - Fifth International Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems. Institute of Electrical and Electronics Engineers Computer Society, Piscataway, NJ 08855-1331, United States, 137-143.

Paper II:

Test Benchmarks – what is the question?

Per Runeson, Mats Skoglund and Emelie Engström

Published in *Proceedings of International Conference on Software Testing Verification and Validation Workshop (ICSTW '08), April 2008*

1 Introduction

"I am taller than you". "My dad is stronger than yours". Kids do not grow very old until they begin benchmarking. They benchmark to impress on their mates and to give themselves a position in the group. But what does the benchmark mean when the child wants to reach the cookies on the top shelf of the larder? Although being the tallest, he might not be tall enough to reach it anyhow, and his father might not be there to lift him up. And if he was, he would not allow his kids to take those cookies anyhow.

In the automotive press, there are lots of benchmarks. Acceleration from 0 to 100 km/h or 0 to 60 mph is a frequently used benchmark. But how often do you accelerate as fast as possible from 0 to 100km/h? Similarly is the power and the torque of the engine benchmarked, but rarely it is noticed whether the power is delivered at revs which are useful in my daily driving or at top revs. And I rarely use more than some 25 kW to run my car, although I have access to hundreds. Furthermore, the EuroNCAP[7] and NTSB[8] do benchmarks on crash resistance and rate car models according to their resistance to the benchmark tests.

When software test researchers benchmark, they use some well specified sets of programs and apply and evaluate their test techniques. The programs are mostly selected based on availability, and sometimes also made available for others; see e.g. the Software artifact Infrastructure Repository[9], although this particular example does not have the ambition of constituting benchmarks [2]. However, before judging whether the benchmarks are useful or not, we should consider what it should be used for. What is the question we want to answer with a benchmark?

2 Uses for test benchmarks

From a practitioner's point of view, the benchmark must focus on the feasibility for the use of the benchmarked techniques and tools in a specific context. "Is this test technique more efficient than the other for my software system?" This is however not a question that can be answered by a single benchmark.

From a researchers' point of view, we have learned that empirical evaluation is good research while blunt assertion is not [1]. Hence, we must have some context in which we may evaluate our techniques and tools. And there is always an issue of relevance; can this be used and useful in software industry?

[7] The European New Car Assessment Programme http://www.euroncap.com

[8] The National Transportation Safety Board http://www.ntsb.gov

[9] http://sir.unl.edu

The benchmarking question involves many degrees of freedom that may impact on the outcome. It is not only the program under test, but its test cases, the defects, its development environment, its development process etc. Hence, the issue of benchmarking is very complex and we find it too ambitious to search benchmarks that mirror all this variation, rather some specific aspects may be studied at a time.

In the automotive domain, where benchmarking takes frequently place, the specific benchmarks may not be of highest relevance, but they are indicators that represent some attributes of the car that a customer may give priority or not. I would choose a car making 0-100 km/h in 5 seconds if I like fast driving (and I can afford it) while for a family car, 0-100 km/h in 10 seconds is sufficient to keep up the daily traffic pace. For crash resistance, I may prefer a five star Euro NCAP rated car before a three star, even though I do not intend to crash it from 64 km/h (40 mph) into a concrete barrier. Instead, the benchmarking procedures have forced car manufacturers to make more crash resistant cars in general in order to fulfill the customer's demands.

In the testing context, benchmarking may be used to indicate specific characteristics (like the acceleration) or be a driving force in a general improvement trend (like crash tests). One of the key issues in finding benchmarks is the representativeness of the benchmark as such. What does it mean in practice that one technique is better than another for a given benchmark?

3 Representativeness

In order to generalize a result from a small set of subjects to a wider population, sampling is applied. For example in national polls or other surveys, a subset from the population are sampled, interviewed about their opinions and conclusions are drawn for the whole population [3]. The sample represents the whole population in a *statistical generalization*. The underlying principles are that the random variation among the subjects is captured in the sample within an acceptable error margin. This is the underlying principle for controlled experimentation.

In qualitative design research, like case studies, the selection is different. The case to be studied is selected to represent e.g. the typical or the special case [4]. The case cannot be generalized to a wider population through statistical analyses. Still one may learn from a specific case and apply the knowledge to another specific case. In case studies you apply *analytical generalization*. In analytical generalization, the case is characterized and compared to other cases to identify patterns which may indicate some general understanding drawn from the specific case.

The search for testing benchmarks may take either way: the *statistical* or the *analytical* approach. The former means defining a population of software programs, sampling from that population and selecting a representative subset which the test techniques may be applied to for evaluation. The statistical approach is desirable but impractical and must hence be excluded. The analytical approach is closer to

what is already done, i.e. using a set of programs, and then generalize the results from the studies analytically.

The analytical approach may be supported by categorization scheme that helps the analytical generalization. Depending on the scope of the evaluated item, benchmarking may be very different, which is elaborated in the next chapter.

Refer to the car crash tests again. Sampling from all possible crashes and repeating a subset in the laboratory would enable calculating a risk factor for a certain car with a specified statistical significance, i.e. statistical generalization. The approach actually used is that some typical situations with frontal and side impact are repeated in the laboratory, i.e. analytical generalization.

4 Variation factors

In the effort for finding typical or special cases or subject programs to be used for benchmarking purposes, many variation factors must be considered. Variation factors may be regarding the program under test, its specifications, the test technique or tool, or the test process or the defects. Factors may be related to the product under test, the test process or the test resources. Below we list some, based on our experience from test research:

Process factors:

- Does the technique require specification documents, e.g. UML diagrams?
- Programming language(s) – is the technique applicable to the programming languages used? What if the there are different languages? If source code is not accessible?
- How many and which type of changes are made between successive releases?
- What is the purpose of the test technique/tool? Test case selection? Test case prioritization?
- Is the technique deterministic, i.e. selects the same test cases independently of who applies it?
- Which types of test are within the scope? Unit test? System test? GUI tests?

Product factors:

- Size and complexity – is the program large and complex enough to be relevant for the real world problem?
- Which type of system is it? Real-time systems vs. batch?
- How is it dependent on code libraries and their changes?
- What size are the test cases, and do they depend on each other?
- Test data – are they complex enough to be relevant for real world problems?
- Defects – are the numbers, types and distribution of defects relevant?

Resource factors:

- Which skills and knowledge do the testers and test designers have?

These variation factors must be taken into account when defining test benchmark programs and processes.

5 Proposal

Based on the considerations above, we propose the following for test benchmarks:

1. Define categories for benchmarked methods to avoid comparing "apples with oranges", e.g. comparing safe test selection methods with unsafe.
2. Look upon benchmarks as selected cases, not representative samples, and interpret benchmarking results accordingly.
3. Define a characterization scheme to capture the relevant degrees of freedom that characterize a test environment.
4. Define not only a set of benchmarking programs, but also the corresponding test cases, defects, execution environment and test processes used.
5. Combine benchmarking results with case studies to analyze both a controlled environment and a real world environment where the interactions between the test technique and its environment can be studied as well.

In summary, the answer is not only a benchmark, but a benchmark in its context. Benchmarking is not aimed at statistical generalization, but analytical. The focus is on the typical or the special situation, not on the "average" situation.

6 References

[1] Shaw M What makes good research in software engineering? *International Journal on Software Tools for Technology Transfer* Springer 4(1):1-7, 2002

[2] Do H, Elbaum S, Rothermel G, Supporting Controlled Experimentation with Testing Techniques: An Infrastructure and its Potential Impact, *Empirical Software Engineering*, 10, 405–435, 2005

[3] Wohlin C, Höst M, Ohlsson MC, Regnell B, Runeson P, Wesslén A *Experimentation in Software Engineering - An Introduction*, Kluwer 2000

[4] Yin R K *Case Study Research. Design and Methods* (3rd edition) London: Sage, 2003

Paper III:

A Qualitative Survey of Regression Testing Practices

Emelie Engström and Per Runeson

Accepted for publication in *11th International Conference on Product Focused Software Development and Process Improvement (PROFES '10), June 2010*

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.

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 indicate 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 RT 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.

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

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?
- 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.

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	Domain	Size
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

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 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 table 1 and table 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, tailored to the domain under investigation. Originally intended for IS development, Zachman [18] proposed that it might be used for developing new approaches to system development. 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 !, 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.

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. Continuosly 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/benifit 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. Consistentcy 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

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 was 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 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 where 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 were 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 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 are 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.

5 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.

6 References

[1] Surveygizmo. http://www.surveygizmo.com, Dec 2009. a web tool for questionnaires and polls.

[2] P. K. Chittimalli and M. J. Harrold. Recomputing coverage information to assist regression testing. IEEE Transactions on Software Engineering, 35(4):452-469, 2009.

[3] E. Engström, P. Runeson, and M. Skoglund. A systematic review on regression test selection techniques. Information and Software Technology, 52(1):14-30, 2010.

[4] E. Engström, P. Runeson, and G. 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, 2010. in press.

[5] Flink. The survey handbook. SAGE Publications, 2nd edition, 2003.

[6] M. Grindal, J. Offutt, and J. Mellin. On the testing maturity of software producing organizations. In Testing: Academia & Industry Conference-Practice And Research Techniques (TAIC/PART), 2006.

[7] IEEE. IEEE standard for software test documentation. IEEE Std(829-1983, Revision),1998.

[8] K. Onoma, W.-T. Tsai, M. H. Poonawala, and H. Suganuma. Regression testing in an industrial environment: Progress is attained by looking backward.

Association for Computing Machinery. Communications of the ACM, 41(5):81-86, 1998.
[9] Robson. Real World Research. Blackwell Publishing, 2nd edition, 2002.
[10] J. Rooksby, M. Rouncefield, and I. Sommerville. Testing in the wild: The social and organisational dimensions of real world practice. Computer Supported Cooperative Work (CSCW), 18(5):559-580, 2009.
[11] G. Rothermel and M. J. Harrold. Analyzing regression test selection techniques. IEEE Transactions on Software Engineering, 22(8):529{552, 1996.
[12] P. Runeson. A survey of unit testing practices. IEEE Software, 23(4):22, 2006.
[13] P. Runeson, C. Andersson, and M. Höost. 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] P. Runeson, P. Beremark, B. Larsson, and E. Lundh. SPIN-syd - a non-profit exchange network. In 1st International Workshop on Software Engineering Net-working Experiences, Joensuu, Finland, 2006.
[15] P. Runeson and M. Höost. Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering, 14(2):131-164, 2009.
[16] M. Skoglund and P. 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] L. White and B. Robinson. Industrial real-time regression testing and analysis using firewalls. Proceedings 20th IEEE International Conference on Software Maintenance, pages 18-27, 2004.
[18] J. A. Zachman. A framework for information systems architecture. *IBM Systems Journal*, 26(3):276-293, 1987.

Paper IV:

An Empirical Evaluation of Regression Testing Based on Fix-cache Recommendations

Emelie Engström, Per Runeson and Greger Wikstrand

Published in *proceedings of International Conference on Software Testing Verification and Validation (ICST '10), April 2010*

Abstract

Background: The fix-cache approach to regression test selection was proposed to identify the most fault-prone files and corresponding test cases through analysis of fixed defect reports. *Aim*: The study aims at evaluating the efficiency of this approach, compared to the previous regression test selection strategy in a major corporation, developing embedded systems. *Method*: We launched a post-hoc case study applying the fix-cache selection method during six iterations of development of a multi-million LOC product. The test case execution was monitored through the test management and defect reporting systems of the company. *Results*: From the observations, we conclude that the fix-cache method is more efficient in four iterations. The difference is statistically significant at $\alpha = 0.05$. *Conclusions*: The new method is significantly more efficient in our case study. The study will be replicated in an environment with better control of the test execution.

1 Introduction

Regression testing is a resource consuming activity in software development. This is particularly true for iterative development approaches, where features are added to existing software in an iterative fashion. Regression testing is performed to ensure that previously functioning software is not corrupted by the changes. Studies indicate that 80% of testing cost is regression testing and more than 50% of software maintenance cost is related to testing [3]

Several techniques for regression test selection are proposed and evaluated. Engström *et al.* recently reviewed the literature in the field [4] and concluded that most of the proposed regression test selection techniques are not feasible to scale up to testing of large complex real time systems. Industry practice on regression testing is mostly based on experience alone, and not on systematic approaches. There is an urgent need to decrease regression testing cost and increase test efficiency in industry.

A pragmatic approach to regression testing is proposed by Wikstrand *et al.* [12]. The basic idea is to link test cases to source files based on information from the test management and defect reporting systems. Test cases are then prioritized with respect to how fault prone their linked files are, if changed. A cache, as proposed by Kim *et al.* [7], is used to monitor which files are fault-prone and fixed in recent iterations. The *fault prediction* effectiveness of the fix-cache method has been shown to be good [12]. However, the efficiency of the *regression testing* based on these recommendations has not been evaluated earlier.

In this paper we report on the first empirical evaluation of test suite efficiency of the fix-cache method. In an industrial setting we compare the efficiency between the traditional manually selected test suites and the test suites recommended by the fix-cache tool. Our results indicate that the tool based selection generates more efficient test suites.

The paper is structured as follows. In Section 2 we briefly present the regression test selection method under study as well as related work. Section 3 presents the design and execution of the evaluation case study. In Section we analyze the validity of the study and we discuss the results and future work in Section 4.

2 Background and related work

The regression test selection algorithm being evaluated in this paper was first described by Wikstrand *et al.* [12]. The algorithm is based on three processes: identifying fault prone source files, linking test cases to source files, and recommending test cases. The company where the study was performed, has a

defect report management system, where affected files are recorded when a defect report is closed. This was crucial to the effectiveness of the described algorithm. The three processes are described below.

a) Identifying fault prone files: When defect reports are closed, the corresponding updated files are marked as hits in a fix-cache as proposed by Kim *et al.* [7]. As recommended in the original paper, the cache size was fixed at 20% of the total number of files. To maintain the size, files were removed from the cache using a least recently used logic.

b) Linking test cases: Also when defect reports are closed, they are traced back to the originating system test case (if any) and marked correspondingly. The test case is thus linked through the closed defect reports to the files which were changed as a result of the fault that was detected when the test case failed.

c) Recommending test cases: When a new test campaign is about to be conducted, the changes on a file level to the product, compared to the previous test iteration, are obtained from the source management system. If a file is both in the fix-cache and has one or more linked test cases, the linked test cases are recommended for execution. References to the linking defect reports are given as a rationale to aid the test leader in deciding whether to follow the recommendation.

Wikstrand *et al.* reported on the precision of the fix-cache. The hit rate, i.e. the number of files with fixes which were already in the cache, of the cache on a week-by-week basis was found to be 50-80% [12], less than the 73-95% reported by Kim *et al.*[7], although in the same magnitude.

Sheriff *et al.* [11] published a study on a similar approach, although with a focus on clusters of files which tend to be changed together. They evaluated the test case selection and found that the methodology proposed test cases, additional to those based on pure file changes, in 50 % of the cases. However, it is not clear from the evaluation whether these test cases actually found more faults or not.

Engström *et al.* published a comprehensive systematic review of all empirical evaluations of regression test selection techniques [4]. They conclude that only 4 out of 15 case studies are conducted in a large scale context, i.e. larger than 100000 LOC, and no more than 1 out of 21 experiments is conducted on large scale artefacts. Software size is not the only criterion for making a study realistic, but the observation calls for more industry evaluations of regression testing methods.

Several studies investigate relationships between fault-proneness and various software metrics, among those lines of code is the most straightforward and most investigated [2]. However, the relation is shown to be logarithmic [8], indicating that smaller classes cause relatively more problems than larger ones. In this study, we only use the characteristic of observed fault proneness as a predictor for future fault proneness.

3 Empirical evaluation

3.1 Research question and method

The aim of the study was primarily to evaluate the efficiency of the fix-cache approach to regression test selection, compared to the previously used regression test selection strategy in a major corporation, developing embedded systems. We refer to efficiency as the number of found faults per selected test case. Our research question is hence:

- Is the fix-cache regression test selection method more efficient than the previously applied experience-based method?

Methods for empirical evaluations include experiments [13] and case studies [10]. Studies of real industrial size are hard to conduct with the level of control required for a formal experiment. Case studies are less controlled, but offer on the other hand a broader spectrum of data to observe. For our evaluation, we have chosen to conduct a case study, in which the data collection and analysis is mostly post hoc.

3.2 Case study setting and results

The case under study is a development project at ST-Ericsson in Lund, Sweden. ST-Ericsson develops platform software and hardware designs for embedded mobile devices. The part of the product under study comprises several million lines of code. It is developed at multiple sites across three continents.

Each week, new increments and fault fixes to a number of the modules in the product are delivered to the main development branch for system and regression testing. In this study, the focus has been on regression test cases from a limited area of system test. The test area in question is representative of the product and tests a cross section of modules and requirements, but we are not able to report any more details about the selected modules for confidentiality reasons.

The fix-cache regression test selection approach was applied during six iterations of regression testing. A list of recommended test cases, based on the method, was delivered to the test department and later followed up by monitoring the test management and defect management databases. Due to lack of control in the case study, not all recommended test cases were executed. We discuss the implications of this in Section 3.3. The actual number of test cases selected and executed are presented in Table 1, as is the total number of executed test cases, based on the ordinary selection method, which mainly is based on fixed test case priorities and test planning heuristics.

We evaluated the fault detection efficiency, defined as:

$$Eff_{\text{det}} = \frac{\#\ faultsfound}{\#testcasesexecuted}$$

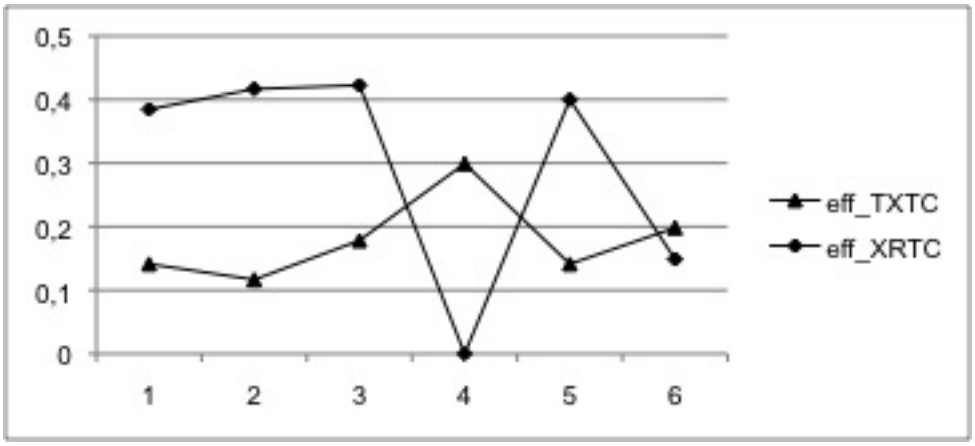

Figure 1 Fault detection efficiency for each iteration

Table 1. Number of test cases for each iteration. RTC = recommended test cases; XRTC = executed recommended test cases; TXTC = total executed test cases

Iteration	1	2	3	4	5	6
RTC	27	41	99	1	11	78
XRTC	13	12	71	1	5	47
TXTC	552	480	1301	906	1203	1317

Table 2. Number of failed tests for each iteration. XRTC = executed recommended test cases; TXTC = total executed test cases

Iteration	1	2	3	4	5	6
XRTC	5	5	30	0	2	7
TXTC	78	56	232	271	170	261

The efficiency for each of the six iterations is reported in Figure 1. Since only one test case was selected in iteration 4, we consider this iteration being an outlier and it is hence excluded from the subsequent analysis. There are probably factors out of the study control that confuse the picture, since the efficiency of the experience-based method is 0.30 for iteration 4, compared to 0.12-0.18 for the other iterations (see Table 3 last row).

The underlying data on selected number of test cases and failed tests are reported in Table 1 and Table 2 respectively. We analyzed the difference between the efficiency of the two approaches using a t-test. There is a significant difference between the two at a 5% significance level t = -3.7033, df = 4.629, p = 0.01602.

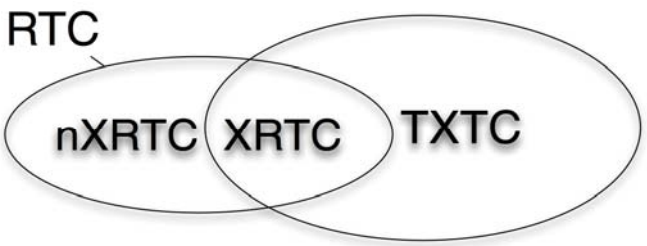

Figure 2. Diagram relating the sets of test cases to each other. RTC = recommended test cases; nXRTC= non-executed recommended test cases; XRTC = executed recommended test cases; TXTC = total executed test cases

3.3 Sensitivity analysis

Since all the recommended test cases were not executed, as reported in Table 1, there is a major threat to the validity of the study that the results are an effect of the properties of the executed set of test cases, rather than the test selection method as such. In order to validate the results, we conducted a sensitivity analysis, calculating theoretical boundaries for the efficiency.

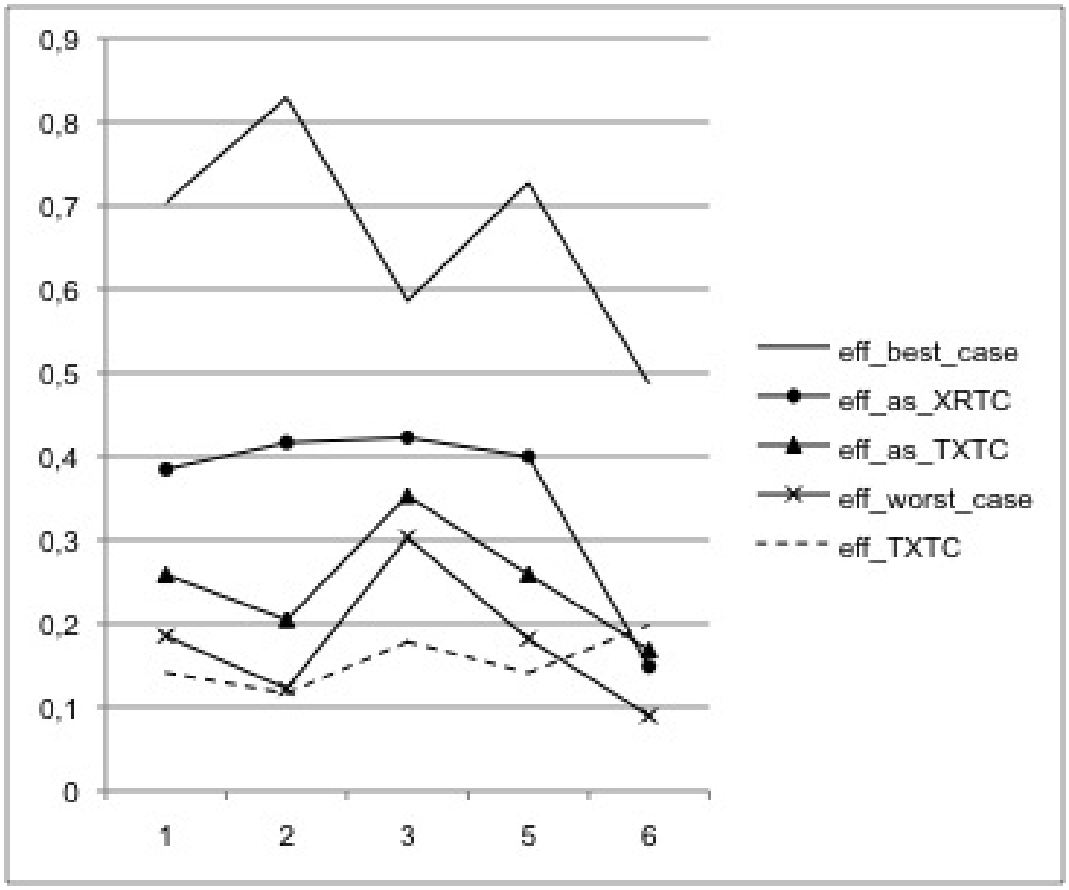

Figure 3. Sensitivity analysis for iterations 1-3, 5 and 6 - share of test cases detecting defects vs. iterations

For the sensitivity analysis, the set of Recommended Test Cases are denoted RTC in Figure 2. The Total number of eXecuted Test Cases (TXTC) does not cover all RTC, and hence only the eXecuted share of the Recommended Test Cases

(XRTC) subset of RTC is executed. We draw our main conclusions based on XRTC only as we do not know the properties of the set of non-eXecuted share of the Recommended Test Cases (nXRTC).

The worst case, i.e. with the lowest efficiency, would be if the test cases of nXRTC all would pass. The best efficiency case would be if all nXRTC fail, although this is not a realistic case. Two further alternative scenarios are a) if the nXRTC are as efficient as the TXTC subset, and b) if the nXRTC are as efficient as the XRTC subset. In the latter case, the RTC efficiency would be exactly the same as the XRTC efficiency, reported above.

These four alternative scenarios are presented in Figure 3 and the data is tabulated in Table 3, using the previously used approach as a reference (eff_TXTC). The worst case is *not* significantly better than the traditional approach ($t = -0.5393$, $df = 5.246$, $p = 0.6118$) while the other two approaches are (eff_best_case: $t = -8.3817$, $df = 4.483$, $p = 0.0006645$; eff_as_TXTC: $t = -2.7125$, $df = 5.661$, $p = 0.0371$).

Table 3. Table 3 Sensitivity analysis using efficiency data for the different approaches over iterations 1- 6, counting iteration 4 as an outlier

Approach	1	2	3	4	5	6
eff_worst_case	0.19	0.12	0.30	N/A	0.18	0,09
eff_best_case	0.70	0.83	0.59	N/A	0.73	0.49
eff_as_TXTC	0.26	0.20	0.35	N/A	0.26	0.17
eff_as_XRTC	0.38	0.42	0.42	N/A	0.40	0.15
eff_TXTC	0.14	0.12	0.18	0.30	0.14	0.20

We conclude from the scenario analysis that the efficiency of the fix-cache test case selection method is not due to the incomplete execution of test cases, but the inherent properties of the method itself.

3.4 Checking assumptions

The fix-cache selection method is based on assumptions concerning *fault churn* and *fault location*. We checked whether these are fulfilled in the studied environment, although with different data sets than the above study.

Fault churn - The fix-cache algorithm is based on the assumption that the faults are not evenly distributed over software modules, and that this distribution is changing over time i.e. there is a *fault churn*. A Pareto-like distribution of faults over modules is statically identified in several studies, e.g. [1,5], while the dynamic behaviour is not studied before, i.e. whether different modules are fault-prone at different occasions.

The assumption was tested by studying post-hoc which modules would have been included in the fix-cache, based on comparing the most fault prone modules in two time periods.

Among modules with at least one fix, the top 20% with the most fixes were selected in each of a three-month period. The share of modules common to the top 20% in the two periods was 66%. A repeated measures ANOVA was conducted on all modules with fixes, with the number of fixes in each of the three-month periods as the dependent variable. The test indicated that the fault distributions were different in the two time periods ($p<.003$), hence the assumption is supported.

Fault location - The other basic assumption is that test cases find faults in the same *fault location*, which sets the upper limit for the method's effectiveness. To test this assumption we analyzed whether test cases, which have failed more than once, lead to fixes in the same modules.

We observed a small number of test cases where fails lead to more than one defect report, causing fixes in the software. 27% of these test cases lead to fixes in the same modules, while the remaining 73% of the test cases lead to fixes in different modules each time. We consider the assumption weakly supported by the studied test cases.

4 Threats to validity

We analyze threats to the validity of the study and report countermeasures taken to reduce them. The definitions follow Wohlin *et al.* [13].

Conclusion validity is concerned with the statistical analyses underpinning the conclusions. The statistical analyses use the robust t-test and in the checking of assumptions, ANOVA. In the sensitivity analysis, we repeat the t-test for each variant, but towards the same reference. Hence, the error rate problem is not apparent.

Internal validity is about the risk for other factors impacting on the relation between what is manipulated and the outcome. The limitation here is that the original set of test cases as well as the small share of selected test cases may not be representative. However, the sensitivity analysis indicates that the conclusions are robust.

Construct validity is concerned with the alignment between what is measured and what is the underlying construct. The test case efficiency measure is only one view of a good regression test selection procedure. The overall defect detection effectiveness is even more important, i.e. the share of defects detected by different test case selection methods. The available data in this case did not allow us to perform such an analysis. Two assumptions for the method was analyzed in Section 3.4 and found supported and weakly supported, respectively, although on a small number of test cases.

External validity is related to generalizability of the results. We have no indications that this environment is unique, but the method should of course be evaluated and tailored to other environments before launching it widely. Its underlying assumptions of Pareto distributed faults is verified by other research, e.g. [1,56] but the dynamic variation over time is not verified in those studies.

5 Discussion and future work

The fix-cache regression test selection technique is a simple, but apparently efficient technique for test case selection. It makes use of information that already is collected and stored in different databases. Setting it into use involves mainly connecting these databases together.

Our empirical evaluation indicates that the technique is more efficient than the previously used technique. The set of test cases that were selected and executed found significantly more defects per test case than the previously used approach did.

Still, there are many questions remaining open. One major question is whether the defect detection effectiveness is better as well. The technique selected a small set of test cases, so the *number* of faults found is very small compared o the number selected by the manual method.

The size of the cache is a factor that impacts on the number of selected test cases. Future evaluations include varying the cache size, and evaluating the efficiency for various sizes of the cache. They should also include data collection to enable analysis of effectiveness measures such as precision and inclusiveness [9]. Other pragmatic strategies, such as random selection of a given percentage should also be applied as a reference.

Replications in other companies and settings are also encouraged to increase the knowledge of the fix-cache regression test selection method.

6 References

[1] C. Andersson and P. Runeson. A replicated quantitative analysis of fault distributions in complex software systems. *IEEE Transactions on Software Engineering*, 33(5):273, 2007.

[2] V. R. Basili and B. T. Perricone. Software errors and complexity: An empirical investigation. *Communications of the ACM*, 27(1):42–53, 1984.

[3] P. K. Chittimalli and M. J. Harrold. Recomputing coverage information to assist regression testing. *IEEE Transactions on Software Engineering*, 35(4):452–469, 2009.

[4] E. Engström, P. Runeson, and M. Skoglund. A systematic review on regression test selection techniques. *Information and Software Technology*, 52(1):14–30, 2010.
[5] N. E. Fenton and N. Ohlsson. Quantitative analysis of faults and failures in a complex software system. *IEEE Transactions on Software Engineering*, 26(8):797–814, 2000.
[6] M. Hamill and K. Goseva-Popstojanova. Common trends in software fault and failure data. IEEE Transactions on Software Engineering, 35(4):484–496, 2009.
[7] S. Kim, T. Zimmermann, E. Whitehead, and A. Zeller. Predicting faults from cached history. *29th International Conference on Software Engineering (ICSE'07)*, pages 489–498, 2007.
[8] Koru, D. Zhang, K. El Emam, and H. Liu. An investigation into the functional form of the size-defect relationship for software modules. *IEEE Transactions on Software Engineering*, 35(2):293–304, 2009.
[9] G. Rothermel and M. Harrold. Analyzing regression test selection techniques. *IEEE Transactions on Software Engineering*, 22(8):529–551, 1996.
[10] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009.
[11] M. Sherriff, M. Lake, and L. Williams. Prioritization of regression tests using singular value decomposition with empirical change records. The *18th IEEE International Symposium on Software Reliability (ISSRE '0*7), pages 81–90, 2007.
[12] G. Wikstrand, R. Feldt, J. Gorantla, W. Zhe, and C. White. Dynamic regression test selection based on a file cache an industrial evaluation. *International Conference on Software Testing Verification and Validation (ICST '09)*, pages 299–302, 2009.
[13] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering: an introduction*. Kluwer, 2000.

Paper V:

Software Product Line Testing - A Systematic Mapping Study

Emelie Engström and Per Runeson

Conditionally accepted for publication in *Journal of Information and Software Technology*

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.

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 [3] 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 for a [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 [67]. 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][72][47][60][52][59] 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.

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.

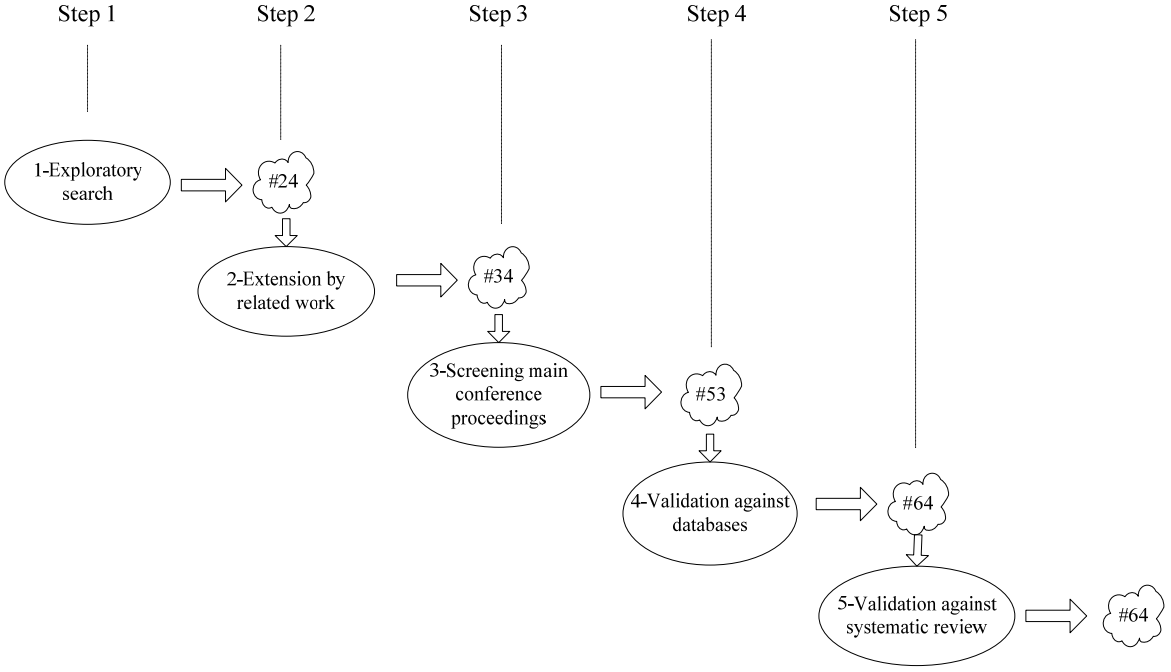

Figure 1 Search for publications on software product line testing

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 [67]. 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 artefacts 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

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 Table1.

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
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 (ICCIT)	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 (ISCSCT)	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

Publication Fora	Type	#
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**

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.

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%

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 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.

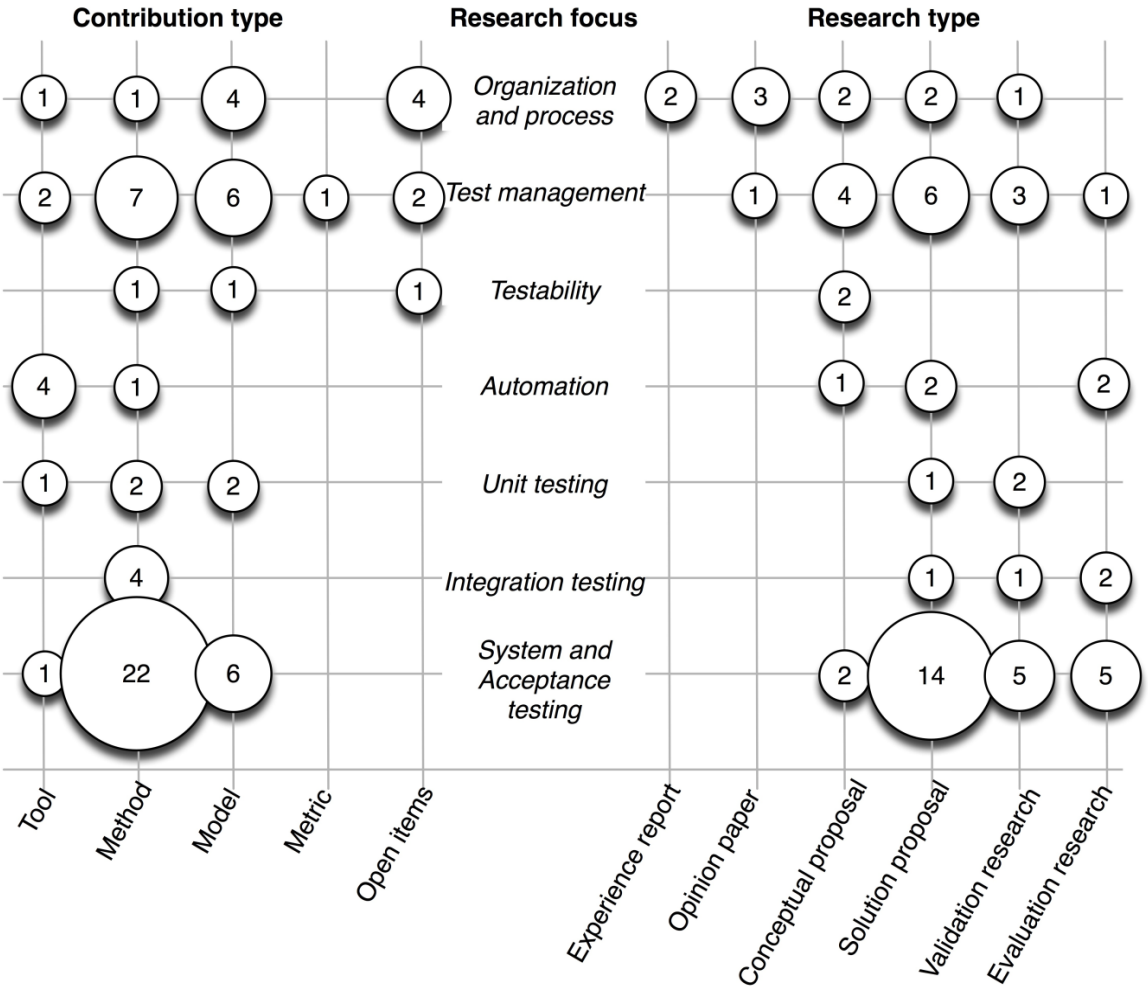

Figure 2 Map of research focus on software product line testing. Research focus on the Y axis; contribution type on the left side of the X axis, and research type on the right side of the X axis.

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.

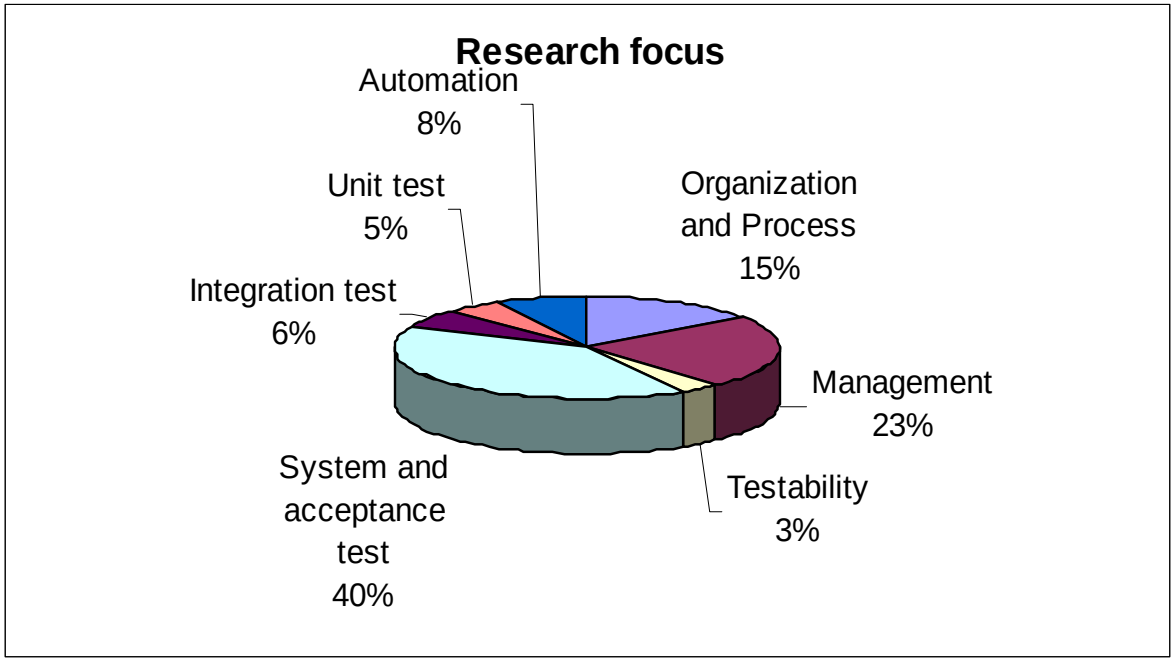

Figure 3 Distribution of research foci

6.1.1 Test Organization 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 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 [73]. 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 [73]. 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 [24]. Ganesan *et al.* [17] 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.* [19] 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]

Table 4. Papers on Test Organization and Process

Author	Title	Paper type	Contribution type
Shaulis (2004) [68]	Salion's Confident Approach to Testing Software Product Lines	Experience report	Tool
Knauber, Hetrick (2005) [32]	Product Line Testing and Product Line Development - variations on a Common Theme	Solution proposal	Method
McGregor (2001)[41]	Structuring Test Assets in a Product Line Effort	Conceptual proposal	Model
Weingärtner (2002) [76]	Product family engineering and testing in the medical domain-validation aspects	Opinion	Model
Ganesan, Knodel, Kolb, Haury, Meier (2007)[17]	Comparing Costs and Benefits of Different Test Strategies for a Software product Line: A study from Testo AG	Validation research	Model
Jin-hua, Qiong, Jing, (2008) [24]	The W-Model for Testing Software Product Lines	Solution Proposal	Model
Kolb, Muthig (2003) [35]	Challenges in Testing Software Product Lines	Opinion paper	Open Items
Tevanlinna, Taina, Kauppinen (2004) [73]	Product Family Testing - a Survey	Opinion paper	Open Items
Kolb, Muthig (2006) [37]	Techniques and Strategies for Testing component-Based Software and Product Lines	Experience Report	Open Items
Ghanam, Park, Maurer (2008) [19]	A Test-Driven Approach to Establishing & Managing Agile Product Lines	Conceptual proposal	Open Items

6.1.2 Test Management

The research on test management contains several proposals and a few evaluated research statements. Tevanlinna proposes a tool, called RITA (fRamework Integration and Testing Application) to support testing of product lines [72]. 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 [43]. 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 [70]. 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.* [9] 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. Kauppinenen *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.* [9] propose application of interaction testing and connect this to the combinatorial coverage criteria.

Table 5. Papers on Test Management

Author	Title	Paper type	Contribution type
Tevanlinna (2004) [72]	Product family testing with RITA	Solution Proposal	Tool
Kolb (2003)[34]	A Risk-Driven Approach for Efficiently Testing Software Product Lines	Solution Proposal	Method
Scheidemann (2006)[70]	Optimizing the selection of representative Configurations in Verification of Evolving Product Lines of Distributed Embedded Systems	Solution Proposal	Method
Gustafsson (2007)[22]	An Approach for Selecting Software Product Line Instances for Testing	Validation Research	Method
McGregor, Im (2007)[43]	The Implications of Variation for Testing in a Software Product Line	Conceptual Proposal	Method
Oster, Schürr, Weisemöller (2008) [57]	Towards Software Product Line Testing using Story Driven Modeling	Conceptual Proposal	Method
Cohen, Dwyer, Shi (2006)[9]	Coverage and Adequacy in Software Product Line Testing	Solution Proposal	Model, Method
Al Dallal, Sorenson (2008) [2]	Testing software assets of framework-based product families during application engineering stage	Validation Research	Model, method, tool
Zeng, Zhang, Rine (2004) [80]	Analysis of Testing Effort by Using Core Assets in Software Product Line Testing	Solution Proposal	Model
Dowie, Gellner, Hanssen, Helferich,	Quality Assurance of Integrated Business Software: An Approach to	Solution Proposal	Model

Author	Title	Paper type	Contribution type
Herzwurm, Schockert (2005) [14]	Testing Software Product Lines		
Jaring, Krikhaar, Bosch (2008) [25]	Modeling Variability and Testability Interaction in Software Product Line Engineering	Evaluation Research	Model
McGregor (2008) [44]	Toward a Fault Model for Software Product Lines	Conceptual Proposal	Model
Kauppinen, Taina, Tevalinna (2004) [29]	Hook and Template Coverage Criteria for Testing Framework-based Software Product Families	Conceptual Proposal	Metric
Denger, Kolb (2006) [11]	Testing and Inspecting Reusable Product Line Components: First Empirical Results	Validation Research	Open Items
Muccini, van der Hoek (2003) [48]	Towards Testing Product Line Architectures	Opinion Paper	Open Items

Al Dallal and Sorenson present a model that focuses on framework testing in application engineering [2]. 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 [14]. 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 [25]. 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 [44] discusses the need for more knowledge on faults likely to appear in a product line 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.

6.1.3 Testability

McGregor discusses testability of software product lines in his technical report. This refers to technical characteristics of the software product that helps testing. 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.

Table 6. Papers on Testability

Author	Title	Paper type	Contribution type
Kolb, Muthig (2006)[36]	Making Testing Product Lines More Efficient by Improving the Testability of Product Line Architectures	Conceptual Proposal	Model, Method
Trew (2004) [74]	What Design Policies Must Testers Demand from Product Line Architects?	Conceptual Proposal	Open Items

6.1.4 System and Acceptance Testing

Table 7. Papers on System and Acceptance Testing

Author	Title	Paper type	Contribution type
Hartmann, Vieira, Ruder (2004)[23]	UML-based approach for validating product lines	Solution Proposal	Tool
Bertolino, Gnesi (2003)[6]	Use Case-based Testing of Product Lines	Solution Proposal	Method
Bertolino, Gnesi (2003)[4]	PLUTO: A test Methodology for product Families	Validation Research	Method
Kamsties, Pohl, Reis, Reuys (2003)[27]	Testing Variabilities in Use case Models	Solution Proposal	Method
Nebut, Pickin, Traon, Jéséquel (2003)[50]	Automated Requirements-based Generation of Test Cases for Product Families	Validation Research	Method
Stephenson, Zhan, Clark, McDermid (2004)[71]	Test Data Generation for Product Lines - A Mutation Testing Approach	Solution Proposal	Method
Geppert, Li, Rössler, Weiss (2004) [20]	Towards Generating Acceptance Tests for Product Lines	Validation Research	Method
Olimpiew, Gomaa (2005) [55]	Model-based Testing for Applications Derived from Software Product Lines	Solution Proposal	Method
Reuys, Kamsties, Pohl, Reis (2005) [64]	Model-Based System Testing of Software Product Families	Evaluation Research	Method
Mishra (2006) [47]	Specification Based Software Product Line Testing: A case study	Solution Proposal	Method
Olimpiew, Gomaa (2006) [53]	Customizable Requirements-based Test Models for Software Product Lines	Evaluation Research	Method
Pohl, Metzger (2006)[60]	Software Product Line Testing	Conceptual Proposal	Method
Reis, Metzger, Pohl (2006)[62]	A Reuse Technique for Performance Testing of Software Product Lines	Evaluation Research	Method
Reuys, Reis, Kamsties, Pohl, (2006) [66]	The ScenTED Method for TestingSoftware Product Lines	Evaluation Research	Method

Author	Title	Paper type	Contribution type
Li, Geppert, Roessler and Weiss (2007) [39]	Reuse Execution Traces to Reduce Testing of Product Lines	Evaluation Research	Method
Bashardoust-Tajali, Corriveau (2008)[8]	On extracting Tests from a Testable Model in the Context of Domain Engineering	Solution Proposal	Method
Kahsai, Roggenbach, Schlingloff (2008)[26]	Specification-based Testing for Software ProductLines	Solution Proposal	Method
Olimpiew, Gomaa (2008)[54]	Model-Based Test Design for Software Product Lines	Solution Proposal	Method
Uzuncaova, Garcia, Khurshid, Batory (2008) [75]	Testing Software Product Lines Using Incremental Test Generation	Validation Research	Method
S Weißleder, D Sokenou, BH Schlingloff (2008) [77]	Reusing State Machines for Automatic Test Generation in Product Lines	Solution Proposal	Method
Dueñas, Mellado, Cerón, Arciniegas, Ruiz, Capilla (2004) [13]	Model driven testing in product family context	Solution Proposal	Model
Nebut, Traon, Jezequel (2006)[52]	System Testing of Product Lines: From Requirements to Test Cases	Validation Research	Model
Olimpiew, Gomaa (2005) [56]	Reusable System Tests for Applications Derived from Software Product Lines	Conceptual Proposal	Model
Kang, Lee, Kim, Lee (2007)[28]	Towards a Formal Framework for Product line Test Development	Solution Proposal	Model, Method
Nebut, Pickin, Traon, Jezequel (2002) [51]	Reusable Test Requirements for UML-Model Product Lines	Solution Proposal	Model, Method
Bertolino, Fantechi, Gnesi, Lami (2006)[5]	Product Line Use Cases: Scenario-Based Specification and Testing of Requirements	Solution Proposal	Model, Method

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 [8].

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.* [5]. 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][50], more comprehensively presented in [52]. They evaluate their approach on a small case study. Reuys *et a*l. 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 [55][54] which was illustrated in a student project [53]. Dueñas *et al.* propose another approach, based on the UML testing profile [13] 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 [8].

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 [20]. 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].

6.1.5 Integration Testing

Table 8. Papers on Integration Testing

Author	Title	Paper type	Contribution type
Reuys, Reis, Kamsties, Pohl, (2006) [66]	The ScenTED Method for Testing Software Product Lines	Evaluation Research	Method
Kishi, Noda (2004)[30]	Design Testing for Product Line Development based on Test Scenarios	Solution Proposal	Method
Li, Weiss, Slye (2007) [40]	Automatic Integration Test Generation from Unit Tests of eXVantage Product Family	Evaluation Research	Method
Reis, Metzer, Pohl (2007)[63]	Integration testing in software product line engineering; A model-Based Technique	Validation Research	Method

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].

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

Table 9. Table 1 Papers on Unit Testing

Author	Title	Paper type	Contribution type
Feng, Liu, Kerridge (2007) [16]	A product line based aspect-oriented generative unit testing approach to building quality components	Validation Research	Method
Reuys, Reis, Kamsties, Pohl, (2003)[65]	Derivation of Domain Test Scenarios from Activity Diagrams	Solution Proposal	Model
Nebut, Fleurey, Traon, Jezequel (2003) [49]	A Requirement-Based Approach to test Product Families	Validation Research	Model, Method, Tool

6.1.7 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 [10] 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.* [18] 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].

Table 10. Papers on Test Automation

Author	Title	Paper type	Contribution type
Knauber, Schneider (2004) [33]	Tracing Variability from Implementation to Test Using Aspect-Oriented Programming	Conceptual Proposal	Tool
Williams (2004)[79]	Test Case Management of Controls Product Line Points of Variability	Solution Proposal	Tool
Condron (2004)[10]	A Domain Approach to Test Automation of Product Lines	Solution Proposal	Tool
Ganesan, Maurer, Ochs, Snoek, Verlage (2005)[18]	Towards Testing Response time of Instances of a web-based Product Line	Evaluation Research	Tool
McGregor, Sodhani, Madhavapeddi (2004)[45]	Testing Variability in a Software Product Line	Evaluation Research	Method

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.

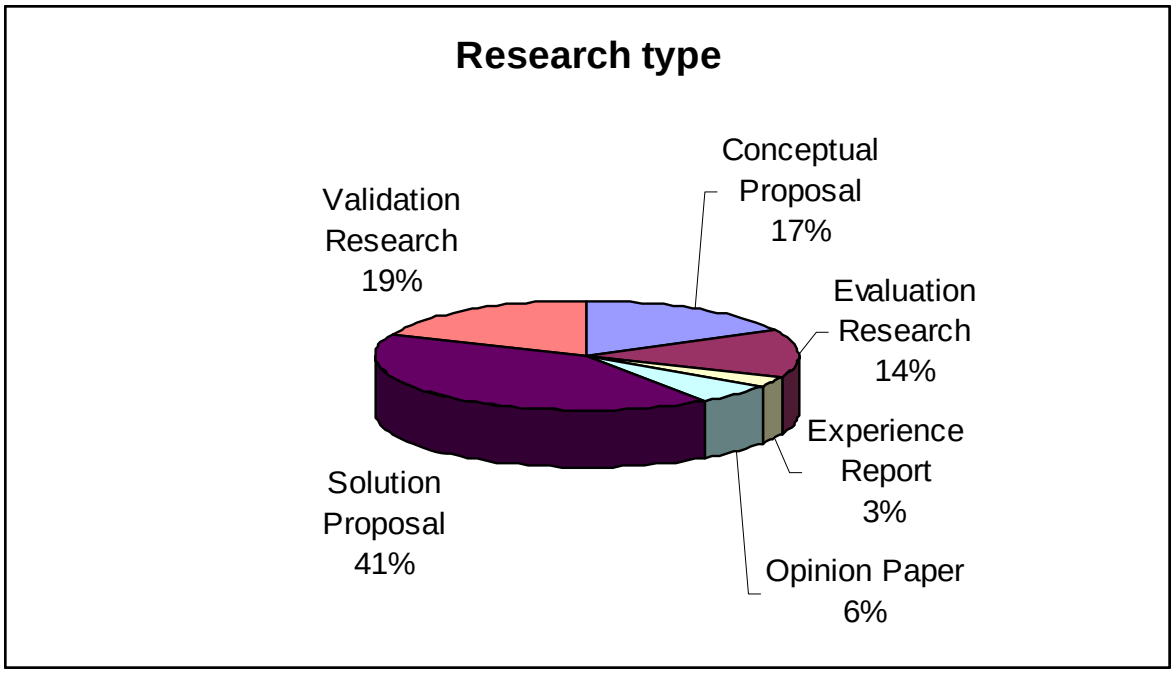

Figure 3 Distribution of Research Type

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 three 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 98. 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 [73], 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 [73].

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.

9 References

[1] W. Afzal, R.Torkar, R. Feldt, A Systematic Mapping Study on Non-Functional Search-Based Software testing, in 20th *International Conference on Software Engineering and Knowledge Engineering (SEKE),* (2008).

[2] J. Al Dallal and P. Sorenson, Testing software assets of framework-based product families during application engineering stage, in *Journal of Software*, Vol 3, No 5 (2008), 11-25, May 2008

[3] J. Bailey, D. Budgen, M. Turner, B. Kitchenham, P. Brereton, S. Linkman, Evidence relating to Object-Oriented Software Design: A survey. *First International Symposium on Empirical Software Engineering and Measurement* (2007).

[4] A. Bertolino and S. Gnesi, PLUTO: A Test Methodology for Product-Families, *5th International Workshop Software Product-Family Engineering*:,Siena, Italy (2003).

[5] A. Bertolino, A. Fantechi, S. Gnesi, G. Lami, Product Line Use Cases: Scenario-Based Specification and Testing of Requirements, Chapter 11 in

*Software Product Lines Research Issues in Engineering and Managemen*t, (Eds.) T. Käkölä and J. C. Duenas, Springer (2006).

[6] A. Bertolino, S. Gnesi, Use Case-based Testing of Product Lines, Proc. *ESEC/FSE*, pp. 355-358, ACM Press.(2003).

[7] J. Bosch, *Design and Use of Software Architectures. Adopting and evolving a product-line approach*. Addison-Wesley (2000)

[8] S. Bashardoust-Tajali, J-P. Corriveau, On Extracting Tests from a Testable Model in the Context of Domain Engineering, *13th IEEE International Conference on Engineering of Complex Computer Systems,* pp.98-107 (2008).

[9] M.B. Cohen, M. B. Dwyer, J. Shi. Coverage and Adequacy in Software Product Line Testing. In: *Proceedings of the ISSTA 2006 Workshop on Role of Software Architecture for Testing and Analysi*s. 53-63, ACM. New York (2006)

[10] C. Condron. A Domain Approach to Test Automation of Product Lines. *International Workshop on Software Product Line Testing.* (2004)

[11] C. Denger, R. Kolb. Testing and Inspecting Reusable Product Line Components: First Empirical Results. *Proceedings 5th International Software Metrics Symposium*. (2006)

[12] O. Dieste, A. Grimán and N. Juristo. (2008) Developing search strategies for detecting relevant experiments. Empirical Software Engineering. DOI: 10.1007/s10664-008-9091-7

[13] J. C. Dueñas, J. Mellado, R. Cerón, J. L. Arciniegas, J. L. Ruiz, R. Capilla. Model driven testing in product family context. *First European Workshop on Model Driven Architecture with Emphasis on Industrial Application.* (2004)

[14] U. Dowie, N. Gellner, S. Hanssen, A. Helferich, G. Herzwurm, S. Schockert. Quality Assurance of Integrated Business Software: An Approach to Testing Software Product Lines. *Proceedings of the 13th European Conference on Information Systems, Information Systems in a Rapidly Changing Economy, (ECIS).* (2005)

[15] E. Engstrom, P. Runeson, M. Skoglund. A systematic review on regression test selection techniques. *Information and Software Technology*, Volume 52, Issue 1, January 2010, Pages 14-30,

[16] Y. Feng, X. Liu, and J. 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* (COMPSAC). (2007).

[17] D. Ganesan, J. Knodel, R. Kolb, U. Haury, G. Meier. Comparing costs and benefits of different test strategies for a software product line: a study from Testo AG, in: *Proceedings of Software Product Line Conference (SPLC)*, 2007.

[18] D. Ganesan, U. Maurer, M. Ochs, B. Snoek, M. Verlage. Towards Testing Response Time of Instances of a Web-based Product Line. In *Proceedings of*

[19] Y. Ghanam, S. Park, and F. A. Maurer. A Test-Driven Approach to Establishing & Managing Agile Product Lines. *The 5th SPLiT Workshop –SPLC 2008*, Ireland.

[20] B. J. Geppert, J. Li, F. Rossler and D. M. Weiss.Towards Generating Acceptance Tests for Product Lines. *8th International Conference on Software Reuse.* Madrid, Spain (2004)

[21] R. L. Glass, I. Vessey, V. Ramesh. Research in Software Engineering: an analysis of the literature, *Information and Software Technology* 44:491-506 (2002).

[22] T. Gustafsson. 2007. An Approach for Selecting Software Product Line Instances for Testing. *International Workshop on Software Product Line Testing. (*2007)

[23] J. Hartmann, M. Vieira, A. Ruder. UML-based Approach for Validating Product Lines. *Intl. Workshop on Software Product Line Testing (SPLiT)*, Avaya Labs Technical Report, pp. 58-64, Boston, USA, August (2004).

[24] L. Jin-hua, L. Qiong and L. Jing. The W-Model for Testing Software Product Lines. *International Symposium on. Computer Science and Computational Technology (ISCSCT).* (2008)

[25] M. Jaring, R.L Krikhaar, J. Bosch. Modeling Variability and Testability Interaction in Software Product Line Engineering. *Seventh International Conference on Composition - Based Software Systems. ICCBSS*. pp.120-129. (2008)

[26] T. Kahsai, M. Roggenbach, B.-H. Schlinglof: Specification-based testing for software product lines. *In Sixth IEEE International Conference on Software Engineering and Formal Methods, SEFM 2008, Cape Town, South Africa, 10-14 November 2008* (2008)

[27] E. Kamsties, K. Pohl, S. Reis and A. Reuys. Testing variabilities in use case models. In *F. van der Linden, Ed., Proceedings of the 5th International Workshop on Software Product-Family Engineering, PFE-5* (Siena, Italy, Nov. 2003), Springer, Heidelberg, 6--18. (2003)

[28] S. Kang, J. Lee, M. Kim, and W Lee. Towards a Formal Framework for Product Line Test Development. In *Proceedings of the 7th IEEE international Conference on Computer and information Technology* (October 16 - 19, 2007). CIT. IEEE Computer Society, Washington, DC, 921-926. (2007)

[29] R. Kauppinen, J. Taina, and A. Tevanlinna. Hook and template coverage criteria for testing framework-based software product families. In *Proceedings of the International Workshop on Software Product Line Testing*, pages 7--12, August (2004).

[30] T. Kishi, and N. Noda. Design Testing for Product Line Development based on Test Scenarios. presented at *Software Product Line Testing Workshop (SPLiT)*, Boston, MA, (2004).

[31] B. A. Kitchenham. Guidelines for performing Systematic Literature reviews in Software Engineering Version 2.3. Technical Report S.o.C.S.a.M. Software Engineering Group, Keele University and Department of Computer Science University of Durham.(2007)

[32] P. Knauber and W. Hetrick. Product Line Testing and Product Line Development - Variations on a Common Theme, *Proceedings of International Workshop on Software Product Line Testing (SPLiT 2005).*(2005)

[33] P. Knauber and J. Schneider. Tracing Variability from Implementation to Test Using Aspect-Oriented Programming. *International Workshop on Software Product Line Testing* SPLiT. (2004).

[34] R. Kolb. A Risk Driven Approach for Efficiently Testing Software Product Lines. *5th GPCE Young. Researches Workshop*, Erfurt, Germany Sep. (2003)

[35] R. Kolb. and D. Muthig, Challenges in Testing Software Product Lines. *In Proceedings of CONQUEST'03,* Nuremberg, Germany, pp. 81--95, September (2003).

[36] R. Kolb and D. Muthig. Making testing product lines more efficient by improving the testability of product line architectures. In *Proceedings of the ISSTA 2006 Workshop on Role of Software Architecture For Testing and Analysis* (Portland, Maine, July 17 - 20, 2006). ROSATEA '06. ACM, New York, NY, 22-27. DOI= http://doi.acm.org/10.1145/1147249.1147252 (2006)

[37] R. Kolb and D. Muthig. Techniques and Strategies for Testing Component-Based Software and Product Lines. Chapter 7 in *Development of Component-Based Information Systems. Advances in Management Information Systems Volume 2 / 2006* , pages 123 - 139 (2006)

[38] B.P Lamancha,. M.P Usaola and M.P Velthius. Software Product Line Testing - A Systematic Review. *4th International Conference on Software and Data Technologies* (*ICSOFT)* p. 23-30. (2009)

[39] J. J. Li, B. Geppert, F.Roessler and D. M. Weiss. Reuse Execution Traces to Reduce Testing of Product Lines. *Proceedings of the International Workshop on Software Product Line Testing* (2007)

[40] J. J. Li, D. M. Weiss and J. H. Slye. Automatic Integration Test Generation from Unit Tests of EXVantage Product Family. *Proceedings of the International Workshop on Software Product Line Testing* (2007).

[41] J. 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] J. D. McGregor, Testing a Software Product Line. Technical Report, CMU/SEI-2001-TR-022, ESC-TR-2001-022.

[43] J. D. McGregor, K. Im. The Implications of Variation for Testing in a Software Product Line. *International Workshop on Software Product Line Testing (SPLiT 2007).*(2007)

[44] J. D. McGregor, Toward a Fault Model for Software Product Lines. In *proceedings Fifth International Workshop on Software Product Line Testing. (SPLiT 2008)* - informatik.fh-mannheim.de. Page 27. (2008)

[45] J. D McGregor, P. Sodhani., S. Madhavapeddi. Testing Variability in a Software Product Line. In: *Proceedings of the International Workshop on Software Product Line Testing*, Avaya Labs, ALR-2004-031, 45–50 (2004)

[46] M. Meyer and A. Lehnerd. The Power of Product Platforms. Free PRess, New York, (1997)

[47] S. Mishra. Specification Based Software Product Line Testing: A case study. *Proceedings of the Concurrency: Specification and Programming Workshop*. Pages 243-254. (2006)

[48] H. Muccini and A. van der Hoek. Towards Testing Product Line Architectures, *Electronic Notes in Theoretical Computer Science* 82 No. 6, (2003).

[49] C. Nebut, F. Fleurey, Y. L. Traon, and J.-M. Jézéquel. A Requirement-based Approach to Test Product Families. In International Workshop on Product Family Engineering (PFE), (2003).

[50] C. Nebut, S. Pickin, Y. Le Traon, and J. M. Jezequel. Automated requirements-based generation of test cases for product families. In *Proceedings 18th IEEE International Conference on Automated Software Engineering* (2003).

[51] C. Nebut, S. Pickin, Y. Le Traon, and J. M. Jezequel. Reusable Test Requirements for UML-Model Product Lines. *International Workshop on Requirements Engineering for Product Lines (REPL)* (2002)

[52] C. Nebut, Y. Le Traon and J. M. Jézéquel. System Testing of Product Lines: From Requirements to Test Cases *Software Product Lines, Research Issues in Engineering and Management*, chapter, pages 447–477. Springer, (2006).

[53] E. M. Olimpiew and H. Gomaa. Customizable requirements based test models for software product lines. *International. Workshop on Software Product Line Testing*, Baltimore, MD , August (2006)

[54] E. M. Olimpiew and H. Gomaa. Model-Based Test Design for Software Product Lines. *International Workshop on Software Product Line Testing (SPLiT 2008).* (2008)

[55] E. M. Olimpiew and H. Gomaa, Model-based testing for applications derived from software product lines, *1st Workshop on Advances in Model-Based Software Testing, A-MOST'05*, ACM Press (2005)

[56] E. M. Olimpiew and H Gomaa. Reusable System Tests for Applications Derived from Software Product Lines, *International Workshop on Software Product Line Testing (SPLiT 2005)*, pp. 8-15. (2005).

[57] S Oster, A Schürr, I Weisemöller. Towards Software Product Line Testing using Story Driven Modeling. *In Proceedings of 6th International Fujaba Days*. Pages 48-55 (2008)

[58] K. Petersen, R. Feldt, S. Mujtaba and M. Mattsson. Systematic Mapping Studies in Software Engineering. *12th International Conference on Evaluation and Assessment in Software Engineering (EASE)*. University of Bari, Italy, 26 - 27 June (2008)

[59] K. Pohl, G. Böckle, and F. van der Linden. *Software Product Line Engineering: Foundations, Principles,and Techniques*, Springer, Heidelberg, August (2005).

[60] K. Pohl and A. Metzger (2006). Software product line testing. *Communications of ACM* 49(12): 78-81.

[61] V. Ramesh, R. L. Glass, I. Vessey. Research in Computer Science: an empirical study, *The Journal of Systems and Science*, 70(1-2):165-176 (2004).

[62] S. Reis, A. Metzger and K.Pohl A Reuse technique for Performance Testing of Software Product Lines. In: *Proc. of the Intl. Workshop on Software Product Line Testing*, Mannheim University of Applied Sciences, Report No. 003.06, 5-10 (2006).

[63] S. Reis, A. Metzger and K.Pohl. Integration Testing in Software Product Line Engineering: A Model-Based Technique. In: Dwyer, M.B.; Lopes, A. (Eds.) *Proceedings Fundamental Approaches to Software Engineering, 10th Intl Conference, FASE 2007*, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007, Braga Portugal, March 24 - April 1, LNCS 4422. (2007).

[64] A. Reuys, E. Kamsties, K. Pohl, and S. Reis. Model-based system testing of software product families. In O. Pastor, and J. Falcao e Cunha, Eds., *Proceedings of the 17th Conference on Advanced Information Systems Engineering, CAiSE* (Porto, Portugal, June 2005), Springer, Heidelberg, 519--534.(2005)

[65] A. Reuys, S.Reis, E.Kamsties and K. 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)*, (2003).

[66] A. Reuys, S. Reis, E. Kamsties and K. Pohl. The ScenTED Method for Testing Software Product Lines. In: *Software Product Lines, pp. 479–520*. Springer, Heidelberg (2006).

[67] P. Runeson and M. Skoglund, Reference-based search strategies in systematic reviews, *13th International Conference on Empirical Assessment & Evaluation in Software Engineering*, Durham University, UK (2009).

[68] C. Shaulis. Salion's Confident Approach to Testing Software Product Lines.In: *Proc. of International conference on Product Line Testing, Boston, Massachusetts, USA* (SPLiT 04). (2004)

[69] M. Shaw, What makes good research in software engineering? *International Journal on Software Tools for Technology Transfer (STTT)*, Springer, 4(1):1433-2779 (2002)

[70] K.D Scheidemann. Optimizing the Selection of Representative Configurations in Verification of Evolving Product Lines of Distributed Embedded Systems. *Proceedings of the 10th International Software Product Line Conference (SPLC'06)*, pp. 75-84. (2006)

[71] Z. Stephenson, Y. Zhan, J. Clark, and J. McDermid. Test Data Generation for Product Lines - A Mutation Testing Approach. In: Nord, R.L. (ed.) *SPLC 2004*. LNCS, vol. 3154, Springer, Heidelberg (2004)

[72] A. Tevanlinna. Product family testing with RITA. *Proceedings of the Eleventh Nordic Workshop on Programming and Software Development Tools and Techniques (NWPER)*. Pages 251-265. (2004)

[73] A. Tevanlinna, J. Taina, R. Kauppinen, Product family testing: a survey. *ACM SIGSOFT Softw. Eng. Notes* 29(2): 12-17.. DOI: 10.1145/979743.979766 (2004)

[74] T. Trew. What Design Policies must Testers Demand from Product Line Architects?, Proc. *Int. Workshop on Software Product Line Testing*, 2004.

[75] E. Uzuncaova, D. Garcia, S. Khurshid, and D. Batory. Testing Software Product Lines Using Incremental Test Generation. In: *ISSRE* (2008)

[76] J. Weingärtner. Product Family Engineering and Testing in the Medical Domain — Validation Aspects. *In Software Product-Family Engineering, 4th International Workshop (PFE)* (Bilbao, Spain, October 3–5 2001), Revised Papers, LNCS 2290/2002. Pages 56-77. (2002)

[77] S Weißleder, D Sokenou, BH Schlingloff. Reusing State Machines for Automatic Test Generation in Product Lines. *1st Workshop on Model-based Testing in Practice* (2008)

[78] R. Wieringa, N. Maiden, N. Mead and C. Rolland et al. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. *Requirements Engineering 11(1)*: 102-107. (2006).

[79] J. J. Williams. Test Case Management of Controls Product Line Points of Variability. *International Workshop on Software Product Line Testing, SPLiT*, (2004)

[80] H Zeng, W Zhang, D Rine. Analysis of Testing Effort by Using Core Assets in Software Product Line Testing. *International Workshop on Software Product Line Testing, SPLiT*, (2004)