Alireza Haghighatkhah

TEST CASE PRIORITIZATION USING BUILD HISTORY AND TEST DISTANCES

AN APPROACH FOR IMPROVING AUTOMOTIVE REGRESSION TESTING IN CONTINUOUS INTEGRATION ENVIRONMENTS
ALIREZA HAGHIGHATKHAH

TEST CASE PRIORITIZATION USING BUILD HISTORY AND TEST DISTANCES
An approach for improving automotive regression testing in continuous integration environments

Academic dissertation to be presented, with the assent of the Doctoral Training Committee of Information Technology and Electrical Engineering of the University of Oulu, for public defence in the Wetteri auditorium (IT115), Linnanmaa, on 31 January 2020, at 12 noon

UNIVERSITY OF OULU, OULU 2020
Haghighatkhah, Alireza, Test case prioritization using build history and test distances. An approach for improving automotive regression testing in continuous integration environments
University of Oulu Graduate School; University of Oulu, Faculty of Information Technology and Electrical Engineering
Acta Univ. Oul. A 739, 2020
University of Oulu, P.O. Box 8000, FI-90014 University of Oulu, Finland

Abstract

Agile software development has become a source of competitive advantage in many industrial sectors, including the automotive domain. Continuous integration (CI) is an agile practice and involves integrating software changes as often and as early as possible. CI in combination with regression testing is considered a good software development practice, because it ensures that recent changes have not negatively impacted the software’s functionality. The execution of a large test suite at each integration (also known as the build) cycle is not realistic. To detect regression faults earlier, one alternative is to permute the test cases within the regression test suite. This is known as test case prioritization (TCP).

This dissertation aims to shorten the build cycle in CI environments by detecting regression faults earlier, allowing software developers to integrate and verify their changes more frequently and continuously. The research was performed in three phases. In the 1st phase, the relevant literature was systematically surveyed to analyze the existing body of knowledge and identify research opportunities. In the 2nd phase, the TCP technique of history-based diversity was developed and subjected to several rounds of improvements and evaluations. In the 3rd phase, results from the previous two phases were synthesized in order to draw conclusions and outline implications.

Our results bring to mind the well-known adage “don’t put all your eggs in one basket”. For the initial stages, when no information about the system under test is available, one should spread the limited testing budget evenly across different parts of the system by running the most dissimilar test cases early in the process (i.e., maximizing diversity as early as possible). For the later stages, when historical build data are available, one should give higher priority to the test cases that have higher historical value (i.e., those that revealed the most failure in the most recent revisions) and that are most distant from those previously prioritized (i.e., cover different areas of the system).

One of the major contributions of this dissertation is the design, implementation, and experimental evaluation of a TCP technique using 12 open-source Java programs. Findings from the experiments provide empirical evidence in support of two previously proposed TCP heuristics – failure recurrence and test diversity – and their combination in CI development environments. Furthermore, results from the systematic mapping studies contribute to the literature by providing a synthesized and structured body of knowledge and a basis for future studies. We conclude that TCP can be performed in CI environments with negligible investment using build history and test distances.

Keywords: agile software development, continuous deployment, continuous integration, regression testing, software engineering, software-intensive systems, test case prioritization
Haghighatkhah, Alireza, Testitapausten priorisointi käyttäen ohjelmiston koonnin historiaa ja testietäisyyttä. Menetelmä regressiotestauksen parantamiseen jatkuvan integroinnin ohjelmistokehityssympäristössä autoteollisuudessa

Oulun yliopiston tutkijakoulu; Oulun yliopisto, Tieto- ja sähkötekniikan tiedekunta

Acta Univ. Oul. A 739, 2020

Oulun yliopisto, PL 8000, 90014 Oulun yliopisto

Tiivistelmä


Tulokset osoittavat, että ei kannata ”laittaa kaikkia munia samaan koriiin”. Testaamisen alku vaiheessa, kun tieota on vain vähän, tulisi testauksen rajallinen budjetti jakaa mahdollisimman erilaisien testitapausten kesken siten, että testaus kohdistuu järjestelmän eri osa-alueisiin (ts. maksimoidaan testauksen monimuotoisuus mahdollisimman aikaisin). Myöhemmässä vaiheissa, kun koontihistoria on saatavilla, tulisi priorisoida niitä testitapaauksia, joilla on suurempi arvo historiassa (eli ne testitapaaukset, joilla löydettiin eniten virheitä viimeisimmässä versiossa) ja jotka ovat mahdollisimman etäisiä verrattuna niihin testitapaauksiin, joita priorisoitiin viimeksi (ts. kattaa ohjelmiston eri osa-alueita).


Asiasanat: jatkuva integrointi, jatkuva ohjelmistokehitys, ketterä ohjelmistokehitys, ohjelmistotuotanto, ohjelmistovaltaiset järjestelmät, regressiotestaus, testitapausten priorisointi
Acknowledgements

Undertaking a PhD abroad has been quite a journey for me, as well as a character-building experience. Along the way, there were many ups and downs, many rejections, and much desperation and uncertainty. Fortunately, I came across the art of SISU – a very Finnish quality one needs to develop to complete a PhD in Oulu (65.0121° N latitude, to be precise). I can now say with all honesty that it was a wonderful experience. This journey gave me the chance to grow both intellectually and personally.

I am grateful for my supervisors – Prof. Markku Oivo as principal supervisor and Prof. Pasi Kuvaja as cosupervisor – who offered me the position and guided me through the journey. I am deeply thankful for their encouraging attitudes and all their advice, for their reviews of my boring manuscripts, for their patience when I spammed their inboxes, and most importantly because they gave me the freedom to explore and learn on my own.

My deep appreciation for the pre-examination of this dissertation goes to Prof. Bram Adams (Polytechnique Montréal, Canada) and Prof. Ville Leppänen (University of Turku, Finland). Their suggestions brought more clarity to this study and helped me improve its quality. I am also very grateful to Prof. Per Runeson (Lund University, Sweden) for his kind willingness to serve as an opponent during my doctoral defense.

I would also like to thank my follow-up group members – Adjunct Prof. Jouni Markkula and Prof. Burak Turhan – whose feedback steered my research in the right direction. I wish to express my gratitude to my coworkers and coauthors, who have worked with me closely over the past five years. I am particularly thankful to Prof. Mika Mäntylä, who guided me in designing the experiments and gave me the confidence to publish our results.

Thanks to all of my wonderful colleagues in the M3S research group. This group provided a first-class environment that enabled me to learn and grow as a PhD student. It has been a privilege to undertake my doctoral studies in such an environment. I would also like to acknowledge the financial support I received from M3S, Tauno Tönningin, and the Nokia Foundation.

I have had the good fortune to meet several people who became close friends over the course of my journey. There are so many that it is impossible to name all of them here, but I truly cherish our friendship and every moment we have spent together.
Lastly, my warmest appreciation goes to my father (the most gentleman I know) and my mother (the most generous person I know). Despite the immense distance that separates us, they have offered loving encouragement and support throughout my journey.

Oulu, October 2019
Alireza Haghighatkhah
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASD</td>
<td>Agile Software Development</td>
</tr>
<tr>
<td>RT</td>
<td>Regression Testing</td>
</tr>
<tr>
<td>CD</td>
<td>Continuous Deployment</td>
</tr>
<tr>
<td>CI</td>
<td>Continuous Integration</td>
</tr>
<tr>
<td>TCP</td>
<td>Test Case Prioritization</td>
</tr>
<tr>
<td>TCS</td>
<td>Test Case Selection</td>
</tr>
<tr>
<td>TSM</td>
<td>Test Suite Minimization</td>
</tr>
<tr>
<td>TSA</td>
<td>Test Suite Augmentation</td>
</tr>
<tr>
<td>SUT</td>
<td>System Under Test</td>
</tr>
<tr>
<td>HBTP</td>
<td>History-based Test Prioritization</td>
</tr>
<tr>
<td>SBTP</td>
<td>Similarity-based Test Prioritization</td>
</tr>
<tr>
<td>HBD</td>
<td>History-based Diversity</td>
</tr>
<tr>
<td>APFD</td>
<td>Average Percentage of Faults Detected</td>
</tr>
<tr>
<td>AMET</td>
<td>Average Method Execution Time</td>
</tr>
<tr>
<td>VDA</td>
<td>Vargha–Delaney A Measure</td>
</tr>
<tr>
<td>NCD</td>
<td>Normalized Compression Distance</td>
</tr>
<tr>
<td>LSH</td>
<td>Locality-sensitive Hashing</td>
</tr>
<tr>
<td>SMS</td>
<td>Systematic Mapping Study</td>
</tr>
<tr>
<td>SLR</td>
<td>Systematic Literature Review</td>
</tr>
<tr>
<td>ASE</td>
<td>Automotive Software Engineering</td>
</tr>
<tr>
<td>MiL</td>
<td>Model-in-the-Loop</td>
</tr>
<tr>
<td>SiL</td>
<td>Software-in-the-Loop</td>
</tr>
<tr>
<td>PiL</td>
<td>Processor-in-the-Loop</td>
</tr>
<tr>
<td>HiL</td>
<td>Hardware-in-the-Loop</td>
</tr>
<tr>
<td>SYSiL</td>
<td>System-in-the-Loop</td>
</tr>
<tr>
<td>ECU</td>
<td>Electronic Control Unit</td>
</tr>
</tbody>
</table>
List of original publications

This dissertation is based on the following articles, which are referred to in the text by their Roman numerals (I–V):


5.2 Threats to validity

5.2.1 Construct validity

5.2.2 Internal validity

5.2.3 External validity

5.2.4 Reliability

5.3 Limitations

5.4 Future work

6 Conclusions

References

Appendices

Original publications
1 Introduction

Software is developed in rapidly changing markets, and software-intensive organizations continuously strive to develop approaches for delivering higher-quality software in shorter time-to-market and at lower costs. The manifesto for agile software development (ASD) emerged in 2001, as a response to such needs (Beck et al., 2001). ASD has since become increasingly mainstream in the industry, and research in the area is now well established (Dingsøyr, Nerur, Balijepally, & Moe, 2012). Few leading organizations have managed to implement ASD to the extent that software is released continuously (Claps, Svensson, & Aurum, 2015). For instance, by May 2011, Amazon was releasing a new version every 11.6 seconds on average, affecting 10,000 hosts simultaneously.¹ This number reached 50 million annual deployments by November 2014, an average of more than one deployment every second.²

This phenomenon is commonly known as continuous deployment (CD) (Farley & Humble, 2010). CD is “the ability to bring valuable product features to customers on demand and at will (deployment) in series or patterns, with the aim of achieving continuous flow (continuity) and in significantly shorter cycles than traditional lead times, from a couple of weeks to days or even hours (speed)” (Rodríguez et al., 2017). CD is neither a novel phenomenon nor a buzz, but realizes early and continuous delivery of valuable software, which is the very first principle of the 12 principles underlying the agile manifesto (Beck et al., 2001). The claimed benefits of CD include shorter time-to-market, increased customer satisfaction, rapid and continuous feedback, rapid innovation, narrower test focus, improved release reliability and quality, and improved developer productivity (Rodríguez et al., 2017).

From the industrial perspective, CD is perceived as a competitive advantage. This is primarily due to the increasing importance and share of software engineering in many industrial sectors. The automotive industry is one such sector; over 80% of its innovation is realized by software-intensive systems (Pretschner, Broy, Kruger, & Stauner, 2007). CD can be implemented to the extent that it brings value to its stakeholders. This can involve automatic deployment of software or simply a set of practices that ensure every

¹Velocity Culture (the unmet challenge in ops): Presentation at O’Reilly Velocity Conference. Santa Clara, CA, June 16, 2011.
change can be deployed, on demand and at will. Once CD is in place, the deployment of software is rather a choice, depending on the underlying business requirements. CD can also be implemented across organizations and among different development parties that aim to improve their inter- and intradevelopment processes. This has particular relevance for the automotive software development process, because it is organized in a highly vertical manner and involves a large chain of suppliers providing different components of the vehicle (Pretschner et al., 2007).

Realizing CD in practice requires investment and development in several areas (Adams & McIntosh, 2016; Claps et al., 2015; Mäntylä, Adams, Khomh, Engström, & Petersen, 2015; Rodríguez et al., 2017). Continuous integration (CI) is an agile practice and arguably the core infrastructure of CD. CI aims to prevent the integration problem known as “integration hell” and automate the build process and verification of changes (Duvall, Matyas, & Glover, 2007). Each integration cycle is also known as a build cycle. The build comprises a set of automated activities and is followed by regression testing (RT). RT involves running a dedicated test suite after each revision to verify that recent changes have not negatively impacted the software’s functionality. RT is widely used in the software industry; it is common to have a dedicated regression test suite that is often run in its entirety (Engström & Runeson, 2010).

Industrial software-intensive systems often comprise many test cases. Each test case includes “a set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement” (IEEE-Std-1012, 2012). The execution of a large regression test suite requires several hours or even days. For example, the JOnAS Java EE middleware requires running 43,024 test cases to verify all of its 16 configurations (Kessis, Ledru, & Vandome, 2005). The software engineering literature has proposed many solutions to improving RT processes (Yoo & Harman, 2012). Test case prioritization (TCP) is one such solution; it involves the ideal ordering of test cases to maximize desirable properties (i.e., early fault detection) (Rothermel, Untch, Chu, & Harrold, 2001). From the fault-detection viewpoint, TCP seems to be a safe approach because it does not eliminate test cases and simply permutes them within the test suite.

A large number of TCP techniques has been proposed (Yoo & Harman, 2012). However, many of them cannot be easily applied in CI development environments, because RT in CI is invoked more frequently and in shorter intervals. Thus, approaches that require exhaustive analysis are overly expensive and inefficient due to the high frequency of changes, which makes the data gathered (e.g., code coverage) by such
approaches imprecise and obsolete (Elbaum, Rothermel, & Penix, 2014). Furthermore, specific requirements in the automotive software industry have resulted in additional challenges for the deployment of TCP techniques. The testing of automotive software occurs in different environments, and the system under test (SUT) varies across these environments (Garousi, Felderer, Karapıçak, & Yılmaz, 2018). Furthermore, implementation details (e.g., source code) may not always be available for the integration and system testing (Caliebe, Herpel, & German, 2012; Lachmann & Schaefer, 2014). This lack of availability can be due to the contractual agreements between the suppliers and customers (i.e., the original equipment manufacturer or another supplier).

Overall, these challenges have left many practitioners with a single alternative: performing RT only occasionally to save resources. This approach gives rise to several concerns, particularly in ASD, in which changes are rapidly introduced and integrated with the system. This development paradigm calls for more frequent and continuous integration of changes, consequently increasing the demand for optimized RT. For these reasons, the automotive software industry is likely to benefit from a TCP technique, which supports developers to detect regression faults earlier and allows frequent integration and verification of the changes in CI development environments. This dissertation addresses this research gap by proposing an alternative approach based on prioritizing test cases within the test suite. Employing the candidate solution in practice, one can expect to detect regression faults earlier and ultimately shorten the RT feedback cycle (i.e., reduce the time required to provide feedback on RT results) for CI of software-intensive systems.

1.1 Objectives and research questions

In view of the research gap identified above, the objective of this dissertation is to shorten the build cycle in CI environments by detecting regression faults earlier, allowing software developers to integrate and verify their changes more frequently and continuously. This objective can be further divided into two parts:

1. Aggregating and analyzing the existing body of knowledge related to the CD of software-intensive systems and automotive software engineering.

2. Designing and evaluating a TCP technique which improves automotive RT by shortening the build feedback cycle in CI environments.

Accordingly, four research questions (RQs) drive the study:
RQ1. What do we know about the CD of software-intensive systems?
RQ2. What do we know about software engineering in the automotive industry?
RQ3. How can we shorten the build feedback cycle in CI environments by prioritizing test cases for the execution?
RQ4. What is the utility of the proposed solution in terms of effectiveness (i.e., average percentage of faults detected) and performance (i.e., method execution time)?

RQ1 and RQ2 address the 1st research objective by aggregating and analyzing the existing knowledge in the literature. RQ3 and RQ4 address the 2nd objective by designing and evaluating a TCP technique which addresses the research problem.

1.2 Overview of the research design

This section introduces the research design, which will be further developed in Chapter 3. The research outcome presented in this dissertation is to be considered applied research (Collis & Hussey, 2013; Nunamaker, Chen, & Purdin, 1990), which “provides a solution to a specific problem by applying knowledge with an aim of improving existing practice or application” (Wohlin & Aurum, 2015). The research was conducted in the following three phases:

- **Phase 1. Exploration:** The objective of this phase was to explore the literature relevant to the subject area and identify research opportunities. During the 1st research phase, the relevant literature was systematically surveyed to analyze the existing knowledge and identify potential research opportunities.
- **Phase 2. Design and Evaluation:** The objective of this phase was to develop and evaluate an approach that addresses the research problem. During the 2nd phase, a TCP technique was developed and evaluated accordingly. The research methodology applied in this phase was experimentation, and the solution underwent several rounds of improvements and evaluations.
- **Phase 3. Research Synthesis:** In this phase, the results from the previous two phases were synthesized in order to draw conclusions and outline implications.

This dissertation consists of the author’s contributions to the research in the 1st and 2nd phases and of five original, peer-reviewed publications (referred to as Papers I–V). Figure 1 presents an overview of the research process, including the research gap, research objectives, research phases, research questions, and publications.
The work presented in this dissertation was conducted in the context of two research programs, that is, DIMECC Need for Speed \(^3\) and ITEA 3 APPSTACLE \(^4\). The scope of the DIMECC Need for Speed research program was to provide the Finnish software-intensive industry with the capability for continuous value delivery. The ITEA 3 APPSTACLE research program aimed to develop open means and interfaces for continuous improvement and deployment of innovative automotive applications and services.

---

**Research Gap**

The automotive software industry is likely to benefit from a TCP technique, which supports developers to detect regression faults earlier and allows for frequent integration and verification of changes in CI development environments.

**Research Objective**

To shorten the build cycle in continuous integration environments by detecting regression faults earlier, allowing software developers to integrate and verify their changes more frequently and continuously.

---

**Phase 1. Exploration**

- RQ1
  - Paper I

**Phase 2. Design and Evaluation**

- RQ2
  - Paper II
  - Paper III

- RQ3
  - Paper IV

- RQ4
  - Paper V

---

**Phase 3. Research Synthesis**

Test Case Prioritization Using Build History and Test Distances: An Approach for Improving Automotive Regression Testing in Continuous Integration Environments

---

**Fig. 1. Phases of the research process underlying the dissertation.**

1.3 **Key findings and implications**

The key results and implications of the dissertation are briefly presented in this section, and a detailed summary of the original publications is presented in Chapter 4.

**Paper I** presents a systematic mapping study that classifies the CD literature. It analyzes the quality of existing research in terms of scientific rigor and industrial

---

\(^3\)DIMECC Need for Speed research program: http://n4s.dimecc.com/en/

\(^4\)ITEA 3 APPSTACLE research program: https://itea3.org/project/appstacle.html
relevance. The study describes 10 recurrent themes that characterize CD, and it aggregates the benefits and challenges reported in the previous studies.

**Paper II** presents a systematic mapping study that classifies and analyzes the literature related to automotive software engineering in order to provide a structured body of knowledge and to identify well-established topics and potential research gaps. This study provides a panoramic overview and a comprehensive summary of the literature on automotive software engineering.

On the basis of the results, a published article (**Paper III**) targets automotive software practitioners and provides a set of recommendations on the basis of existing literature. The recommendations are related to various subjects associated with contemporary software engineering in the automotive industry.

**Paper IV** presents an experiment in which a TCP technique was developed and evaluated. The experiment was conducted on six open-source Java projects, each of which included several CI builds over a long time period (between 2013 and 2016). Overall, from a pool of 12,995 analyzed builds, 1,330 faulty builds were identified and investigated.

The developed technique combines two previously proposed heuristics and is called history-based diversity (HBD). Test diversity, the first heuristic, is a classical heuristic in software testing (e.g., Chen, Leung, and Mak (2004); Jiang, Zhang, Chan, and Tse (2009); Ledru, Petrenko, and Boroday (2009); Leon and Podgurski (2003); Thomas, Hemmati, Hassan, and Blostein (2014)). The core idea is that test cases that capture the same faults tend to be more similar to each other, and test cases that capture different faults tend to be more dissimilar. Failure recurrence, the second heuristic, has a particular relevance in CI development environments (e.g., Elbaum et al. (2014); Hemmati, Fang, Mäntylä, and Adams (2017); Kim and Porter (2002); Marijan, Gottlieb, and Sen (2013); Srikanth, Cashman, and Cohen (2016)). This heuristic implies that test cases that failed in previous builds have a much higher probability of failing again in the current build.

HBD automatically analyzes archived CI build logs and collects information about the regression test suite (i.e., the list of executed test cases and their verdicts) for each revision. Using a simple function, HBD aggregates test cases into several clusters on the basis of their historical value (i.e., revealed most failure in most recent revisions). Thereafter, HBD iteratively reprioritizes the intracluster tests on the basis of their distance (dissimilarity) to the set of already prioritized tests. When archived CI build logs are not yet available, HBD aims to maximize diversity as early as possible by
giving higher priority to a test that has maximum distance (is most dissimilar) to the set of already prioritized tests.

Paper IV shows that TCP in CI environments can be performed with negligible investment using build history and that its effectiveness can be further improved by considering distances among the test cases. From this study, we learned that similarity-based TCP (SBTP) has great potential and can be applied in different contexts, particularly during initial testing, when no information about the SUT is available. SBTP can also be applied in a complementary fashion and combined with other TCP techniques, as demonstrated previously.

Paper V investigates SBTP more closely. This paper implements five different techniques previously proposed in the literature and presents an experiment conducted with the Defects4j dataset (Just, Jalali, & Ernst, 2014), which contains 395 faults from six open-source Java programs (between 2006 and 2015). The findings show that running the most dissimilar test cases early in the testing process largely improves the test suite’s fault-detection capability. Of the five implementations investigated, no technique was found to be superior with respect to effectiveness. SBTP using locality-sensitive hashing (LSH) was slightly less effective than other investigated techniques, but it seems to be more practical when performance becomes critical.

Implications: Taken together, the results of the original publications have important academic and practical implications. From the academic perspective, Papers I-III provide researchers with a synthesized and structured body of knowledge and a basis for future studies. Papers IV and V elaborate on the design, implementation, and evaluation of the TCP technique of HBD. These studies provide further empirical evidence in support of two previously proposed TCP heuristics—failure recurrence and test diversity—and their combination in CI development environments.

From a practitioner’s perspective, Papers I-III provide a synthesized database and structured repository that may facilitate access to and comprehension of state-of-the-art knowledge. Papers IV and V suggest that TCP can be performed in CI environments with negligible investment using build history and test distances. For the initial stages, when historical data are scarce or not yet available, one should spread the limited testing budget evenly across different parts of the system by running the most dissimilar test cases early in the process (i.e., maximizing diversity as early as possible). For the later stages, when archived build logs are available, one should give higher priority to the test cases that have higher historical value (i.e., those that revealed the most failure in the
most recent revisions) and are most distant from those previously prioritized (i.e., cover different areas of the system).

1.4 Dissertation structure

Figure 2 illustrates the overall structure of the dissertation, which is organized into six chapters. Following this introductory chapter, Chapter 2 reviews the relevant literature and positions the dissertation within the broader research arena. Chapter 3 presents the research design, including the research phases and detailed information about their implementation. Chapter 4 summarizes the original publications included in this dissertation. Chapter 5 discusses the results and their implications, including a discussion of the threats to validity and the limitations of the dissertation, and it identifies future research opportunities. Finally, Chapter 6 offers concluding remarks. Appendix 1 provides further technical information about the TCP techniques applied in Papers IV and V.
Fig. 2. Dissertation outline.
2 Background

The work presented in this dissertation is positioned in the areas of software-intensive systems, software development processes, CI, and RT. Section 2.1 presents a general model and definition of RT improvement techniques. Section 2.2 discusses challenges that arise as the result of the intersection between CI and RT. Section 2.3 summarizes the existing studies relevant to TCP in CI development environments. Section 2.4 describes the automotive software testing process and the characteristics of that process that make the deployment of RT improvement techniques challenging. Finally, Section 2.5 summarizes the research gaps that motivated the dissertation.

2.1 Regression testing techniques

Figure 3 depicts a general model of RT techniques. Let $P$ be a program, $P'$ be a modified version of the program, and $T$ be a test suite developed for $P$. In the transition from $P$ to $P'$, a previously verified behavior of $P$ may have become faulty in $P'$. RT seeks to validate $P'$ to ensure that recent changes to the system have not negatively affected any previously verified functionalities.

During RT, one can employ several techniques in practice. Test suite minimization (TSM) seeks to identify and permanently eliminate obsolete or redundant test cases from the test suite. Test case selection (TCS) aims to select only the subset of test cases affected by the recent changes. TCP involves the ideal ordering of test cases to maximize desirable properties (i.e., early fault detection). Test suite augmentation (TSA) aims to identify newly added source code and to generate new test cases accordingly.

There is a large body of research on RT, with a great deal of pioneering work published in the 1990s (Harrold, Gupta, & Soffa, 1993; Rothermel & Harrold, 1996; Rothermel, Harrold, Ostrin, & Hong, 1998; Rothermel, Untch, Chu, & Harrold, 1999). Several secondary studies of the RT literature were previously identified by a tertiary study conducted by Garousi and Mäntylä (2016). The most seminal survey of the subject area was conducted by Yoo and Harman (2012), who summarized a detailed analysis of 159 papers and discussed several open problems for future work. Despite the large body of knowledge about RT and the many academic advancements in the field, there is relatively little evidence regarding the practical application of RT techniques in industrial settings (Orso & Rothermel, 2014; Yoo & Harman, 2012). The industry
practice seems to be based mainly on experience rather than on any systematic approach to RT (Engström & Runeson, 2010). To ensure that RT techniques are useful and easily adopted, contextual factors related to enterprise testing environments must be considered (Do, 2016).

2.2 Regression testing and continuous integration

CI is an agile practice in which software developers integrate their work frequently, often multiple times per day. Each integration cycle (also known as the build cycle) includes a set of automated activities for detecting integration errors as early as possible. During each build cycle, the CI server automatically retrieves recent changes from the version control system and invokes a set of preconfigured automated activities, such as compiling, packaging, and RT. Thereafter, the CI server notifies the interested stakeholders of the relevant information (e.g., build status).

Once CI is in place, a reliable build is always available with minimal manual overhead (Duvall et al., 2007). Considered a good practice in software development, RT in combination with CI ensures that recent changes do not introduce any regression
into the system. However, this combination presents the software industry with major challenges. These challenges have been addressed in the literature, often in isolation. One prevalent challenge in this context is the long build cycle, which is due to the exhaustive retesting of the system at each revision. To deal with this challenge, researchers have proposed several TCS and TCP techniques, which are designed to be particularly applicable in CI development environments (Ekelund & Engstrom, 2015; Elbaum et al., 2014; Hemmati et al., 2017; Knauss et al., 2015; Marijan et al., 2013; Memon et al., 2017; Spieker, Gotlieb, Marijan, & Mossige, 2017; Srikanth et al., 2016; Strandberg, Sundmark, Afzal, Ostrand, & Weyuker, 2016; Yoo, Nilsson, & Harman, 2011). These techniques aim to select and prioritize test cases for the execution and ultimately shorten the build feedback cycle (i.e., reduce the time required to provide feedback on RT results).

Test parallelization & virtualization is another alternative for rapidly executing a large regression test suite (Bell & Kaiser, 2014; Bell, Melski, Dattatreya, & Kaiser, 2015; Gambi, Gorla, & Zeller, 2017; Parsa, Ashraf, Truscan, & Porres, 2016). Test parallelization aims to identify the dependencies among test cases and build a test plan to execute them in parallel environments in a cost-efficient manner. Test virtualization, on the other hand, aims to minimize the overhead of isolating each test using a container sharing common resources.

Research has demonstrated that the long build cycle can also be due to the exhaustive rebuilding of the system at each revision (Celik, Knaust, Milicevic, & Gligoric, 2016; Jendele, Schwenk, Cremarenco, Janicijevic, & Rybalkin, 2019; Shi, Thummalapenta, Lahiri, Bjornier, & Czerwonka, 2017; Vakilian, Sauciuc, Morgenthaler, & Mirrokni, 2015). The unnecessary dependencies that arise during the build process reduce software modularity and increase the load on the CI server by triggering extra builds and test executions. These studies have proposed approaches to reducing the build time through techniques such as optimal test placement, decomposing build targets into smaller units, and lazy and partial retrieval of the dependent software libraries that are needed during the execution of a build target.

Another challenge is to continuously keep the regression test suite updated as the system evolves and new changes are integrated. Several TSA techniques have been proposed in the literature (e.g., Campos, Arcuri, Fraser, and Abreu (2014); Campos, Fraser, Arcuri, and Abreu (2015); Mirzaaghaei, Pastore, and Pezzè (2014); Robinson,

\[\text{The information provided in this section is not exhaustive, but rather indicative of RT challenges in CI development environments.}\]
Ernst, Perkins, Augustine, and Li (2011). These techniques aim to automatically update the regression test suite. Fault localization is another challenge in CI development environments (Jiang, Zhang, Chan, Tse, & Chen, 2012; Laghari, Murgia, & Demeyer, 2016; Ziftci & Reardon, 2017). These studies have proposed approaches to automatically identifying the locations of regression faults in a program while minimizing manual human intervention.

2.3 Continuous test case prioritization

TCP has been widely investigated in the literature, and a large number of techniques have been proposed. There are several secondary studies of the TCP literature (e.g., Catal and Mishra (2013); Rosero, Gómez, and Rodríguez (2016); Singh, Kaur, Suri, and Singhal (2012); Yoo and Harman (2012)). The most recent systematic literature review on TCP techniques was conducted by Khatibsyarbini, Isa, Jawawi, and Tumeng (2018). The authors identified 80 papers and classified existing TCP approaches into nine categories. They concluded that TCP approaches are still generally open for improvement; each approach has specific potential values, advantages, and limitations.

TCP can be achieved using a variety of techniques. These techniques rely on different heuristics (e.g., maximizing coverage) and require different input resources. For instance, TCP can be performed using coverage information. This technique is based on the hypothesis that maximizing coverage of the system as early as possible could lead to earlier detection of regression faults. The coverage information can be on different levels (e.g., statement, branch, or method coverage) and can be obtained using dynamic analysis of the program execution or static analysis of the test and source code.

TCP can also be model based and achieved using specification models of the SUT. These approaches require extra information that may not be commonly available in all development environments. For instance, Korel, Koutsogiannakis, and Tahat (2007) proposed various model-based TCP techniques, in which different information about the system model (i.e., the original and modified system models together with information collected during execution of the modified model on the test suite) and its behavior is used to perform TCP. Overall, the deployment of a TCP technique is context dependent and influenced by the availability of input resources.

Several TCP techniques have been proposed for CI development environments. The study by Kim and Porter (2002) was one of the first to apply history-based test prioritization (HBTP). A seminal work in the area is the study conducted by Elbaum...
et al. (2014). They performed an empirical study on a large dataset obtained from Google and presented novel TCS and TCP techniques. The proposed approaches are based on the use of time windows to track how recently test suites were executed and revealed failures. This information was utilized to select test suites to be executed during presubmit testing and to prioritize test suites that must be performed during postsubmit testing.

Several other TCP techniques that rely on historical data have been proposed. For instance, Marijan et al. (2013) proposed a TCP technique that relies on previous failure knowledge, test execution time, and domain-specific heuristics to compute the test priority using a weighted function. Strandberg et al. (2016) presented an experience report and proposed an automated tool that aimed to combine priorities of multiple factors associated with test cases. Spieker et al. (2017) applied reinforcement learning to select and prioritize test cases according to their duration, previous last execution, and failure history. Findings from Srikanth et al. (2016) suggest that ordering build acceptance tests can significantly impact the efficiency of testing and that the use of historical data is a good heuristic for test prioritization.

A recent experiment conducted by Hemmati et al. (2017) investigated the effectiveness of three blackbox TCP techniques on Firefox before and after the transition to rapid releases. Their results suggest that HBTP is far more effective than other comparable techniques in rapid releases, although the same conclusion does not hold in traditional development environments. They argued that this superiority may be due to the recency of historical knowledge, which is a better explanation for its effectiveness in a rapid-release environment than other changes in the development process.

The existing HBTP techniques exhibit minor differences. All the proposed techniques share a common heuristic and rely on previous failure knowledge. Failure recurrence is the name of the heuristic applied. The underlying idea is that *test cases that failed in previous builds have a much higher probability of failing again in the current build.* HBTP is easy to implement, but it has inherent limitations. For instance, not all regression faults can be captured effectively if there has been no previous failure, as with newly added test cases or those that have not previously revealed any failure. To effectively detect regression faults, HBTP should be applied in combination with other TCP techniques.

Coverage-based TCP has been extensively studied (see the latest systematic literature review on TCP by Khatibsyarbini et al. (2018)). A common heuristic is maximizing coverage as early as possible. This can be achieved by assigning a higher rank to the test
cases that cover a part of the system that has not been examined earlier by other test cases (Rothermel et al., 2001). To increase the likelihood of detecting faults, particularly when coverage information is unavailable, one can aim to maximize diversity as early as possible. The core idea is to spread the testing budget evenly across different parts of the system by performing a diverse set of test cases.

To maximize diversity as early as possible, one needs to measure similarities among the test cases. The concept of similarity measurement is of interest for many applications (e.g., comparative genomics, music classification, plagiarism detection). The degree to which two objects share characteristics is called similarity, and the degree to which they differ is termed distance.

As a classical heuristic, test diversity has been widely applied in the literature. For instance, in adaptive random testing, diversity has been applied to distribute the test cases evenly over the input domain (Chen et al., 2004; Jiang et al., 2009). This heuristic has also been applied to perform TCP (Hemmati et al., 2017; Ledru et al., 2009; Leon & Podgurski, 2003; Thomas et al., 2014) and TCS (Cartaxo, Machado, & Neto, 2011; Feldt, Poulding, Clark, & Yoo, 2016; Hemmati, Arcuri, & Briand, 2010, 2013; Hemmati & Briand, 2010). Test diversity has been applied in several other areas, that is, TSM (Cruciani, Miranda, Verdecchia, & Bertolino, 2019), TSA (Ma, Wu, & Chen, 2018; Shahbazi & Miller, 2016), and mutation testing (Shin, Yoo, & Bae, 2017), and has been used to improve the test case automation process by maximizing reuse (Flemström, Pasqualina, Daniel, Wasif, & Bohlin, 2018).

The underlying idea is that test cases that capture the same faults tend to be more similar to each other, and test cases that capture different faults tend to be more dissimilar. The implication for TCP is that a higher priority must be assigned to test cases that are most distant from those already prioritized. To realize this in practice, one needs to measure distances among test cases using a specific method and then leverage this information to perform TCP. This can be realized using a variety of techniques, including metrics that measure the similarity between test cases and algorithms that maximize diversity within the test suite as early as possible. This procedure can be automated and realized by using different properties associated with test cases, such as the source code (Ledru, Petenko, Boroday, & Mandran, 2012), test input and output (Henard, Papadakis, Harman, Jia, & Le Traon, 2016), topic models extracted from test scripts (Thomas et al., 2014), and even English documents of manual test cases (Hemmati et al., 2017).


2.4 Testing automotive software

Over 80% of innovations in the automotive industry are now realized by software-intensive systems (Pretschner et al., 2007). This is evidenced by premium vehicles, which contain onboard software that is about 100 MLOC (million lines of code) in magnitude and that is deployed in 50 to 120 embedded microcontrollers (Ebert & Favaro, 2017). Software in automotive systems is spread across various application areas, and it ranges from low-level control software to advanced driver-assistance and infotainment systems. Due to the increasing presence of software in automotive systems, it is critical to identify and fix faults as early in the development process as possible.

Figure 4 presents the V development process, including X-in-the-loop testing, which is widely applied in embedded software and the automotive industry (Garousi et al., 2018). The testing of automotive software occurs in various stages and environments, that is, model-in-the-loop (MiL), software-in-the-loop (SiL), processor-in-the-loop (PiL), hardware-in-the-loop (HiL), and system-in-the-loop (SYSiL). These environments have different characteristics, and the SUT varies across them.

![Fig. 4. The V development process, including X-in-the-loop testing (adapted from Garousi et al. (2018)).](image)

During the initial stages and at the MiL level, the control function and the environment model are developed and tested using a system modeling notation. The SUT at the MiL level is a system model that is often developed using Matlab Simulink and Stateflow. Once the model is developed and verified using MiL, the next stage is SiL, in which the control function is converted to actual source code (partly manually and...
partly using automatic approaches). Testing at the SiL level occurs on the source code (i.e., C or C++), which can be deployed and executed on the target platform simulated using the environment model. The next stage is PiL, which goes beyond the simulated environment and involves the deployment and testing of binary code deployed on a target (external) processor. Thereafter, at the HiL level, the control software is fully integrated with the control system, that is, the electronic control unit (ECU). The objective of HiL testing is to verify the integration of hardware and software in a more realistic environment. The final step is SYSiL, in which testing occurs on the physical embedded system.

RT can occur at all stages and can be performed either automatically or manually. This can be a resource-intensive process, because test suites in the automotive industry are often automatically generated and include a large number of test cases (Caliebe et al., 2012; Lachmann & Schaefer, 2014). This can be a challenge, particularly in agile development environments, where changes are rapidly introduced and integrated into the system. As the automotive industry moves towards the agile development paradigm, there will be more frequent and continuous integration of changes. The adoption of ASD in the automotive industry is evidenced by the many case studies and experience reports in the literature (Ebert & Favaro, 2017; Hohl, Münch, Schneider, & Stupperich, 2016; Katumba & Knauss, 2014; Manhart & Schneider, 2004; McCaffery, Pikkarainen, & Richardson, 2008). The adoption of ASD consequently increases the demand for optimized RT. However, deploying existing RT improvement techniques in the automotive industry is challenging for several reasons.

The testing of automotive software occurs in different environments, that is, MiL, SiL, PiL, HiL, and SYSiL, and the SUT varies across them. Furthermore, automotive development performs in a distributed environment and involves a large chain of suppliers (Pretschner et al., 2007). Software components are often developed by the suppliers and delivered to the original equipment manufacturer or another supplier for the integration process. Due to contractual agreements, implementation details (e.g., source code) are often unavailable for the integration and system testing (Caliebe et al., 2012; Lachmann & Schaefer, 2014). Thus, a candidate RT improvement technique should be independent of the SUT and its implementation details.

A number of studies have investigated RT improvement in the automotive industry. Caliebe et al. (2012) argued that automatically generated test suites are typically large and have to be reduced by applying TCS and TCP techniques. The authors proposed a TCS and TCP technique that relies on the dependencies between the components.
of embedded systems that are derived from the system architecture and requirement specification. Vöst and Wagner (2016) proposed a TCS technique in order to support CI in the automotive industry. The approach does not require access to the source code and relies on the signal traces during test execution within the network of ECUs. Arrieta et al. (2018) proposed a TCS technique for the Simulink models that relies on three different antipatterns based on output signals and two similarity metrics (input- and output-based similarities). Using test coverage and test diversity, Matinnejad, Nejati, Briand, and Bruckmann (2019) proposed an approach to automatically generating and prioritizing test cases for the Simulink models.

2.5 Research gap

On the basis of the previous sections, three observations can be made:

1. The intersection of CI and RT creates major challenges for software development. In CI development environments, RT is invoked more frequently and in shorter intervals. To maintain the development momentum, (semi)automated techniques are required to make the RT both efficient (i.e., reduced execution time) and effective (i.e., earlier detection of regression faults).

2. A large number of RT improvement techniques have been proposed. However, many of them cannot be easily applied in CI development environments. Traditional RT improvement techniques often rely on code-coverage criteria and require exhaustive analysis. These approaches are inefficient and overly expensive due to the high frequency of changes in CI environments, which makes the data gathered (e.g., code coverage) by such approaches imprecise and obsolete (Elbaum et al., 2014).

3. The deployment of an RT improvement technique is context dependent and influenced by the availability of input resources. The specific characteristics of the automotive software industry should be considered when deploying an RT improvement technique in an automotive project. The testing of automotive software occurs in different environments, that is, MiL, SiL, PiL, HiL, and SYSiL, and the SUT varies across them. Furthermore, depending on the contractual agreement, implementation details (e.g., source code) are not always available for the integration and system testing processes.

Given these observations, a candidate TCP technique must fulfill three requirements:
1. **Blackbox**: The technique does not require access to the actual source code of the SUT or any representation of it (e.g., specification models). This makes the candidate solution applicable during integration and system testing, when implementation details are not always available due to contractual agreements.

2. **Static**: The technique does not require the analysis of the SUT’s execution behavior. Thus, a blackbox, static TCP technique requires neither access to the implementation details of the SUT nor an analysis of its execution behavior. The only required input is information about the test cases, which is already encoded in the regression test suite. This makes such techniques widely applicable and represents an improvement over other TCP techniques, which often require access to the runtime behavior, specification models, or the source code of the SUT.

3. **Minimal**: The technique can be applied with negligible investment in terms of input data. This makes the candidate solution applicable even during the early stages of the development process, when no information about the SUT is available at hand.

   Taking these requirements into account, this dissertation develops and evaluates a TCP technique that aims to improve automotive RT in CI environments. Employing the candidate solution in practice, one can expect to detect regression faults earlier and ultimately shorten the RT feedback cycle (i.e., reduce the time required to provide feedback on RT results) for CI of software-intensive systems.
3 Research design

The work presented in this dissertation was carried out in three phases: exploration, design and evaluation, and synthesis. Figure 5 illustrates the overall research design process. The 1st phase (exploration) included two systematic mapping studies, in which we surveyed the literature relevant to the subject area. The objective was to analyze the existing knowledge and identify research opportunities. During the 2nd phase (design and evaluation), a TCP technique was implemented and underwent several rounds of improvement and evaluation. This phase included two experimental studies, and its goal was to design a solution that meets the requirements identified above and addresses the research problem. During the 3rd phase (research synthesis), the results from the previous phases were synthesized in order to draw conclusions and outline implications.

![Fig. 5. Research design process.](image)

3.1 Research design decisions and rationale

The research design process involves several phases and decision points. To guide software engineering researchers during the research design, Wohlin and Aurum (2015)
proposed a conceptual framework that includes three design phases and eight decision points. The strategy phase involves a plan that guides the researcher’s execution of subsequent research phases, that is, the tactical and operational phases. The strategy phase includes decision-making regarding the outcome, logic, purpose, and approach of the research. The next phase is tactical; it involves decision-making regarding how to operationalize the research activities with respect to research process and methodology. Finally, the operational phase involves decision-making regarding actions that will be taken during implementation, including data collection methods and data analysis techniques. Figure 6 shows the research phases and design decisions made during the research design process.

The research outcome (decision point 1) of this dissertation is to be considered applied research (Collis & Hussey, 2013; Nunamaker et al., 1990), which “provides a solution to a specific problem by applying knowledge with an aim of improving existing practice or application” (Wohlin & Aurum, 2015). This is in contrast with basic research, which emphasizes understanding the problem and has knowledge as its sole outcome. The research logic (decision point 2) of this dissertation is inductive. Inductive research is based on a bottom-up approach that induces generalizations from specific observations, whereas deductive research takes a top-down approach, working from general conclusions to specific observations. This dissertation is based on inductive arguments built upon results from the literature analysis and experiments on 12 open-source Java projects.
The purpose of research can be classified as exploratory, descriptive, explanatory and evaluative research (Collis & Hussey, 2013). The research (decision point 3) presented in this dissertation is exploratory and evaluative. The research problem was identified using exploratory research by surveying the literature, whereas the utility of the proposed solution was investigated using evaluative research.

The research approach (decision point 4) taken in this dissertation was pragmatism, which “adopts an engineering approach to research; it values practical knowledge over abstract knowledge, and uses whatever methods are appropriate to obtain it” (Easterbrook, Singer, Storey, & Damian, 2008). The pragmatic perspective contrasts with the underlying views of other research approaches, such as positivism, interpretivism, and critical analysis. The research process (decision point 5) combined qualitative and quantitative research (known as the mixed approach). Qualitative research was applied during the exploration phase, whereas quantitative research was applied during the design and evaluation phase. Two research methodologies (decision point 6), a systematic mapping study and a technology-oriented experiment, were employed in this dissertation.

The data collection method (decision point 7) was archival research, which involves the systematic collection, analysis, and interpretation of archived data. During the exploration phase, published articles were extracted from bibliographic databases. During the design and evaluation phase, the data were extracted by analyzing software repositories (i.e., build logs from the CI server, change list and source code from version control systems) from several open-source projects. The data analysis method (decision point 8) combined qualitative and quantitative analysis. For the systematic mapping studies, qualitative thematic analysis was performed to analyze the data extracted from the research articles. For the experimental studies, descriptive analysis and nonparametric statistical analysis were performed to investigate the TCP techniques with respect to two variables: effectiveness and performance.

3.2 Research implementation

This section describes the research methodologies applied during the 1st phase (exploration) and 2nd phase (design and evaluation) of the research.
3.2.1 Systematic mapping study

Following the evidence-based software engineering paradigm (Kitchenham, Dyba, & Jorgensen, 2004), many literature reviews have been published. The main difference between a systematic literature review (SLR) and a systematic mapping study (SMS) is that the former aims to “identify best practice with respect to specific procedures, technologies, methods, or tools by aggregating information from comparative studies”, whereas the latter targets broad research areas and focuses on “classification and thematic analysis of literature” (Kitchenham, Budgen, & Brereton, 2011).

To analyze the existing literature and identify potential research opportunities, we applied an SMS during the exploration phase. We followed the guidelines for performing SLRs proposed by Kitchenham and Charters (2007) and adapted these guidelines to an SMS, as recommended by Petersen, Vakkalanka, and Kuzniarz (2015). Figure 7 illustrates the research process used to conduct the research for Papers I and II.

![Fig. 7. Systematic mapping study: Research procedure.](image)

During the planning stage (steps 1–4), a review protocol was developed and used as a guideline for the rest of the research process. The review protocol included the research objective and research questions, search strategy, selection criteria, selection process, data extraction properties, and the procedure for the data extraction, analysis, and interpretation of the results. To identify relevant research (step 5), we conducted searches on several bibliographic databases, which were selected based on their coverage of the software engineering literature. Paper II also included a domain-specific digital
library, that is, the Society of Automotive Engineers (also known as SAE), which indexes a wide range of articles related to automotive engineering.

The primary study selection (step 6) was performed using the selection criteria that were explicitly defined in the review protocol. To minimize the subjectivity of the selection process, each step was conducted by a pair of researchers, and disagreements were resolved by joint discussion between the coauthors involved in the review. The data extraction phase (step 7) was performed individually using the data extraction form previously defined in the review protocol. Thereafter, data extraction forms were integrated and conflicts were identified by cross-checking the results. To resolve these disagreements, a third reviewer joined the original two reviewers, and the rationale for resolution was discussed with the remaining reviewers in meetings.

During the data analysis and synthesis phase (step 8), we performed a qualitative analysis of the primary studies. Paper I had a narrow scope and included an in-depth analysis of 50 primary studies using both inductive and deductive coding. The analysis procedure followed the thematic synthesis guideline proposed by Cruzes and Dybå (2011). Paper II had a broader scope and included a qualitative analysis of 679 articles. Both Papers I and II included analysis and classification of the primary studies with respect to the investigated research subjects, research methodological properties, and potential research gaps. Paper I further analyzed the quality of existing studies with respect to scientific rigor and industrial relevance using the framework proposed by Ivarsson and Gorschek (2011).

3.2.2 Technology-oriented experiment

The need for experimental software engineering was emphasized in the mid-1980s by Basili, Selby, and Hutchens (1986). Experimentation is a form of empirical enquiry in which a researcher manipulates one or more variables, controls all other variables, and investigates the effect of the manipulation (Wohlin et al., 2012). In human-oriented experiments, humans apply different treatments to objects, whereas in technology-oriented experiments, different technical treatments are applied to different objects (Wohlin et al., 2012).

During the design and evaluation phase, we conducted experiments in which we applied various TCP techniques to different versions of several open-source Java programs. Figure 8 illustrates the research process that was applied in Papers IV and V.
Figures 8 and 7 share a common research procedure and are different only with respect to the operation phase.

The research design and evaluation phase started with the replication of experiments conducted by Ledru et al. (2012) and Hemmati et al. (2017). The purpose of the replication study was to understand what has been proposed previously and how it can be applied in other contexts as well as to identify potential opportunities for improvement. Thereafter, a research goal was identified, an initial pilot study was conducted, and a research protocol was developed (steps 1–4). The research protocol included research objectives, initial research questions, and the operation plan for the implementation of the experiments.

The data extraction and preparation was performed in step 5. The data for Paper IV were extracted using the TravisTorrent database (Beller, Gousios, & Zaidman, 2017). From a pool of 12,995 analyzed builds, 1,330 faulty builds were identified and investigated. For each faulty build, we analyzed archived CI build logs in order to extract information about previous execution of the regression test suite (i.e., the list of executed test cases and their verdicts). Furthermore, we automatically extracted the changes list and source code behind each revision from the Github repository.

The extracted data were then imported to a database management system for further processing. A similar procedure was applied in Paper V. However, the latter procedure was easier, because the data for Paper V were extracted from a dataset called Defects4j.
(Just et al., 2014), which contains 395 real faults from six open-source Java programs. The data used in both Papers IV and V are publicly available.

The experiment was implemented in step 6 using Apache Groovy, a modern programming language based on Java virtual machine. The implementation underwent several iterations and validations before the actual experiment was conducted. For each faulty build, we performed TCP using different techniques. To compare the investigated TCP techniques, we measured both effectiveness and performance.

To assess effectiveness, we used an average percentage of faults detected (APFD) metric that was originally introduced by Rothermel et al. (2001) and is widely applied in the literature (Khatibsyarbini et al., 2018). Let $T$ be an ordered test suite containing $n$ test cases and $F$ be a set of $m$ faults detected by $T$; then $TF_i$ indicates the number of test cases executed in $T$ before capturing fault $i$. APFD is defined as follows:

$$\text{APFD} = 100 \ast \left(1 - \frac{TF_1 + TF_2 + \ldots + TF_m}{nm} + \frac{1}{2n}\right).$$

The performance was measured using average method execution time (AMET) in seconds and included both the preparation time (e.g., similarity measurement) and the TCP algorithm itself. The experiment was performed in step 7, and the measurements for each observation (i.e., each analyzed version of program) were recorded for the next step.

During the analysis and interpretation phase (step 8), we retrieved the recorded information and conducted a statistical analysis following the guideline proposed by Arcuri and Briand (2011). The objective of our analysis was to identify potential differences between the TCP techniques under investigation and to determine their magnitude. A Mann–Whitney $U$-test, which is a nonparametric significance test, was used to determine whether the difference between two techniques was statistically significant, using $p < 0.05$ as the significance threshold. The null hypothesis of this test indicated that there was no significant difference between the effectiveness of the techniques under evaluation. This test was selected because the studied data may not follow a normal distribution.

The Mann–Whitney $U$-test identifies differences between techniques but not the magnitude of such differences. Thus, we used a Vargha–Delaney A (VDA) measure, which is a nonparametric effect size. A VDA measure is a number between 0 and 1. When $VDA(x,y) = 0.5$, it indicates that the two techniques are equal. When $VDA(x,
\( y > 0.5 \), it means \( x \) outperformed \( y \), and vice versa. Furthermore, when comparing TCP techniques, we visualized the distribution of APFDs using violin plots.
4 Original research papers

This section describes the publications included in the dissertation. The publications are all ranked in the JUFO system, which is a classification of publication channels created by the Finnish scientific community to support the quality assessment of academic research. Papers I, II, and IV were published in the *Journal of Systems and Software* (JUFO 3). Paper III was published in *IEEE Software* (JUFO 2) and Paper V in the *International Conference on Product-Focused Software Process Improvement* (JUFO 1).

The author of this dissertation had a major involvement and the lead role in every phase of the research performed in Papers II-V. The author of this dissertation led the writing of Papers II-V, and the coauthors provided feedback by reviewing the manuscripts. The 2nd and 3rd coauthors of Paper II contributed to the data collection and analysis phase and were involved in writing Appendix B of the manuscript. In Paper I, the author of this dissertation was involved in all research phases (i.e., study design, writing the review protocol, data collection and analysis, and interpretation of the results) and made a major contribution to the manuscript writing (i.e., Sections 2, 5.2, 5.7, 5.8, 6, and 8).

A brief summary of original publications is presented in Table 1. The following subsections provide an in-depth description of each publication, including its rationale, key results, contributions, and role in the dissertation.

---

7The Finnish Publication Forum is also known as JUFO, which is derived from the Finnish term “Julkaisufoo- rum”. The JUFO classification has four levels: 0 = publication channels that do not meet minimum standards; 1 = basic; 2 = leading; 3 = top.
<table>
<thead>
<tr>
<th>Publications</th>
<th>Purpose</th>
<th>Key Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publications</td>
<td>Purpose</td>
<td>Key Results</td>
</tr>
<tr>
<td>--------------</td>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>Paper IV. Haghhighatkhah, A., Mäntylä, M., Oivo, M., &amp; Kuvaja, P. (2018). Test Prioritization in Continuous Integration Environments. Journal of Systems and Software, 146, 80-98.</td>
<td>Detecting regression faults earlier to allow software developers to integrate and verify their changes more frequently and continuously.</td>
<td>For the initial stages, when historical data are not yet available or scarce, one should spread the limited testing budget evenly across different parts of the system by running the most dissimilar test cases early in the process. For the later stages, when archived build logs are available, one should give higher priority to the test cases that have higher historical value (i.e., those that revealed the most failure in the most recent revisions) and are most distant from those previously executed (i.e., cover different areas of the system).</td>
</tr>
</tbody>
</table>
Table 1. High-level summary of original publications.

<table>
<thead>
<tr>
<th>Publications</th>
<th>Purpose</th>
<th>Key Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paper V. Haghighatkhah, A., Mäntylä, M., Oivo, M., &amp; Kuvaja, P. (2018). Test Case Prioritization Using Test Similarities. In International Conference on Product-Focused Software Process Improvement (pp. 243-259). Lecture Notes in Computer Science, vol. 11271. Springer, Cham.</td>
<td>Investigation of whether SBTP is more effective at finding defects than random permutation, and identification of the implementations that lead to better results.</td>
<td>To effectively consume a limited testing budget, one should spread it evenly across different parts of the system by running the most dissimilar test cases early in the process. Of the five SBTP implementations investigated, no technique was found to be superior with respect to effectiveness. LSH seems to be more practical when performance becomes critical.</td>
</tr>
</tbody>
</table>

4.1 Paper I

Rationale: Paper I presents a systematic mapping study in which we aggregated, analyzed, and classified the literature on CD. The objective was to provide an overview of the state of the art on CD, characterize the phenomenon, aggregate reported benefits and challenges, investigate the scientific evidence in the existing studies, and identify areas suitable for further research.

Results: An in-depth analysis of the primary studies revealed 10 recurring themes that characterize CD. These themes were (1) fast and frequent release, (2) flexible product design and architecture, (3) continuous testing and quality assurance, (4) automation of the entire deployment pipeline, (5) configuration management, (6) customer involvement, (7) continuous and rapid experimentation, (8) postdeployment activities, (9) agile and lean software development, and (10) organizational factors, including integrated corporate functions, transparency, and an innovative and experimental organizational culture. The benefits of applying CD include shorter time-to-market, increased customer satisfaction, rapid and continuous feedback, rapid innovation, narrower test focus, improved release reliability and quality, and improved developer productivity.
The challenges were related to the organization-level changes resulting from the move towards CD, customers’ unwillingness to receive continuous releases, increased quality assurance efforts, and difficulties in applying CD in the embedded domain. The evaluation of the primary studies’ scientific rigor and industry relevance identified relevant limitations to the scientific quality of the existing literature. We concluded that the research area is still in its infancy and that an increase in the number and improvement of the rigor of empirical studies is required.

**Contribution:** The study presents a logical classification of the CD literature in order to provide researchers with a structured body of knowledge on which to base future studies. The study also informs practitioners about the main factors (i.e., recurring themes) they should consider when deciding whether to migrate to CD, reflecting on the potential benefits of CD, and anticipating the CD-related challenges they might face.

**Thesis Relevance:** In the context of this dissertation, the contribution of Paper I is a body of knowledge related to CD, in which “continuous testing and quality assurance” are recognized as essential elements of CD. To realize CD in practice, one needs to minimize manual overhead by automating the entire deployment process. This requires an automated regression test suite that can be invoked continuously (i.e., frequently and in short intervals, as advocated by CD). The execution of a large test suite at each revision is not realistic. This is simply because the testing budget is often limited in industry, and RT needs to make the most of limited resources. Thus, one must effectively minimize, select, and prioritize test cases for the execution.

### 4.2 Paper II

**Rationale:** Paper II presents a systematic mapping study in which we aggregated, analyzed, and classified the literature on automotive software engineering (ASE). The review includes 679 articles from multiple research subareas, published between 1990 and 2015. The primary studies were analyzed and classified with respect to five different dimensions. Furthermore, potential research gaps and recommendations for future research are presented.

**Results:** Fourteen recurring subjects in the literature were identified. These subjects are diverse, which indicates the complexity and breadth of ASE research. Three subjects, namely system/software architecture and design, system/software testing, and software reuse, were identified as the most frequently addressed areas. Evaluation research and solution proposals were prevalent in the literature (43% and 22% of the primary...
studies, respectively). However, rigorous validation and comparative studies, which guide industry in the selection of existing solutions in practice, were less common. Surprisingly, a large number of primary studies were not grounded in any research methodology (51% of primary studies). Furthermore, many of the case studies in our review did not report appropriate details as recommended by previous guidelines. These methodological limitations hinder the future replication, generalization, and aggregation of evidence in the context of ASE.

The majority of existing contributions (50% overall) took the form of technical constructs, that is, frameworks, methods, or techniques. However, practitioner-oriented guidelines and concrete tools were less common (4.5% and 11% of the contributions, respectively). From the practical point of view, these figures raise questions regarding the applicability of the existing solutions (i.e., which approach is more appropriate in what context, and why). Thus, rigorous scientific studies that conduct systematic exploration, validation, and evaluation of the existing knowledge are required to provide guidelines for the selection of existing solutions, technologies, and practices.

**Contribution:** From the academic point of view, this systematic mapping study provides researchers with a structured body of knowledge and a basis for future studies. From the perspective of practitioners, the synthesized and structured body of knowledge may facilitate access to and comprehension of state-of-the-art knowledge of ASE.

**Thesis Relevance:** In the context of this dissertation, the contribution of Paper II was a panoramic overview and comprehensive summary of the ASE literature. We observed that, despite progress in the research area, research themes indicated by the ASE roadmap (Pretschner et al., 2007) seem to be both valid and relevant, representing opportunities for future research. Two of these research themes were “improving development processes” and “improving quality and reliability”. These themes have a special relevance to the problem area investigated in this dissertation.

### 4.3 Paper III

**Rationale:** Paper III presents a study in which we aggregated and synthesized information from academic and practitioner-oriented literature, analyzed the material, and developed a set of practitioner-oriented recommendations. Papers II and III share a common set of primary studies but differ in terms of scope and target audience. Paper II mainly targets the ASE research community and communicates through an academic
channel. However, Paper III presents results in a more accessible and shorter format, targeting ASE practitioners.

**Results:** The results include a set of recommendations concerning six different areas: software architecture, software quality assurance and testing, requirement engineering, agile practices, variability management, and software reuse. Apart from these recommendations, we provided public access to the full list of analyzed studies and data extraction results.

**Contribution:** The main contribution is a thorough characterization and analysis of the state of knowledge and a set of practitioner-oriented recommendations.

**Thesis Relevance:** An important challenge in automotive system integration is the effective synthesis and deployment of software functions. The syntactical and technological integration-related challenges are often addressed during the design phase by using a standardized architecture (Giese, Neumann, Niggemann, & Schätz, 2007). However, more demanding integration problems, such as semantic integration and system dependability, are handled late in the integration process (Giese et al., 2007). ASD advocates frequent and continuous integration of the changes. To perform integration as often and as early as possible, practitioners should automate the integration procedure and invest in virtualized integration platforms. This makes CI possible during the development process and minimizes dependency on other parts of the system that may be unavailable at the time. CI systems enable more frequent and continuous integration of changes. Consequently, there is an increased need for optimized RT in the automotive industry.

### 4.4 Paper IV

**Rationale:** Paper IV reports an experiment in which we examined TCP in CI environments. The objective of this study was to catch regression faults earlier, allowing software developers to integrate and verify their changes more frequently and continuously. To achieve this, we investigated six open-source projects, each of which included several CI builds over a long time period (a total of 1,330 faulty builds between 2013 and 2016).

**Results:** Only a small proportion of tests failed with our subjects (less than 11% in four projects, and 3–52% overall). This indeed underlines the importance of TCP in CI environments, in which RT is performed more frequently and continuously. Furthermore, the majority of regression faults (57–97%) among all investigated projects can be
captured using only previous failure knowledge. This implies that previous failure knowledge has strong predictive power in CI environments and can be used to effectively prioritize tests. HBTP does not necessarily require a large amount of historical data, and its effectiveness improves to a certain degree with a larger history interval. Even with the last verdict (current−1), improvement (VDA: 0.53–0.82) in terms of APFD was observed in all investigated projects in comparison to random ordering.

In our experiments, we performed diversity-based TCP using three different similarity metrics, that is, Manhattan, normalized compression distance (NCD), and NCD multisets. Appendix 1 provides further technical information about the similarity metrics and TCP techniques applied. Our results suggest that diversity-based TCP can be used effectively during the early stages, in which no historical data are available (VDA: 0.68–0.91, improvement observed using NCD multisets) or can be combined with HBTP to improve its effectiveness (VDA: 0.51–0.73, using NCD multisets). Among the investigated TCP techniques, we found that HBD using NCD multisets is superior in terms of effectiveness but introduces relatively high overhead in terms of method execution time.

**Contribution:** Taken together, the results imply that HBTP can be employed in practice with negligible investment and that its effectiveness can be further improved by considering distances (dissimilarities) among the tests. The main contribution of this study is the design, implementation, and evaluation of a TCP technique, that is, HBD. The study further contributes to the literature by providing empirical evidence in support of two previously proposed TCP heuristics (i.e., failure recurrence and test diversity) and their combination in CI development environments.

**Thesis Relevance:** In the context of this dissertation, the contribution of Paper IV is the design and evaluation of a TCP technique that can be applied in CI development environments. HBD is a blackbox and static TCP technique. HBD relies only on the information already encoded in the test suite, which makes it a minimal technique that can be applied without any precollected input data. HBD can also automatically analyze archived CI build logs (when they are available) and incorporate historical data while performing TCP.
4.5 Paper V

**Rationale:** Paper V reports an experiment in which we investigated various SBTP techniques. The objective was to determine whether SBTP is more effective at finding defects than random permutation and to identify the SBTP implementations that lead to better results. To achieve our objective, we implemented five different techniques from the literature and conducted an experiment using the Defects4J dataset.

**Results:** Test suites ordered by SBTP were largely more effective at finding defects than random permutation (VDA: 0.76–0.99, observed using NCD across all subjects). This indicates that running the most dissimilar test cases early in the testing process (maximizing the diversity) improves the test suite’s fault-detection capability. This was also verified by our sanity check, in which the reverse approach was applied (VDA: 0.03–0.34). Of the five SBTP implementations investigated, no technique was found to be superior with respect to effectiveness. SBTP using LSH was slightly less effective than other SBTP techniques (VDA: 0.38 observed in comparison to NCD), but it seems to be more practical when performance becomes critical.

**Contribution:** Taken together, these results bring to mind the well-known adage “don’t put all your eggs in one basket”. To effectively consume a limited testing budget, one should spread it evenly across different parts of the system by running the most dissimilar test cases early in the process. This study contributes to the literature by providing empirical evidence in support of test diversity and its impact on TCP.

**Thesis Relevance:** From the previous study (Paper IV), we learned that SBTP requires minimal information and has a broad range of potential applications, particularly during initial testing, when no information about the SUT is available. SBTP is an interesting approach when code instrumentation is impossible or too costly. SBTP can also be applied in a complementary fashion and combined with other TCP techniques, as demonstrated in Paper IV. Paper V investigates SBTP more closely. In the context of this dissertation, the contribution of Paper V is its empirical evaluation of the effectiveness and performance of various SBTP techniques.

---

8In the context of Paper V, we intentionally used the term "SBTP" rather than "diversity-based TCP". As noted in Section 5.4 (future work), SBTP has a broader application area and can be applied to realize different exploitation strategies, that is, intensification and diversification.
5 Discussion

This chapter presents a synthesis of the dissertation’s main findings. The summary of results, along with their implications, is presented in Section 5.1. Section 5.2 discusses threats to the validity of the results and the countermeasures taken during the research process. The limitations of the dissertation are discussed in Section 5.3. Finally, Section 5.4 identifies and discusses potential areas of investigation for future research.

5.1 Summary of the results

The objective of this dissertation was to shorten the build cycle in CI environments by detecting regression faults earlier, allowing software developers to integrate and verify their changes more frequently and continuously. Accordingly, four research questions were posed, and they were addressed in three research phases.

The research started with the exploration phase, in which we investigated the literature using two systematic mapping studies to identify potential research opportunities. The need for these reviews emerged in the context of the DIMECC Need for Speed and ITEA 3 APPSTACLE research programs. Findings from this research phase suggest that CD of software-intensive systems requires development and investment in several areas. CD calls for more frequent and continuous integration of changes, which consequently increases the demand for optimized RT.

The execution of a large test suite at each revision is not realistic. This is simply due to the fact that the testing budget is often limited in industry, and RT needs to make the most of limited resources. Thus, one must effectively minimize, select, and prioritize test cases for the execution. From the fault-detection viewpoint, TCP seems to be a safe approach, because it does not eliminate test cases and simply permutes them within the test suite. The goal of permutation is to detect defects as early as possible. In this way, any partial execution of the regression test suite detects the maximum number of defects for the given budget.

During the design and evaluation research phase, we reviewed all existing secondary studies on RT, which were identified by a tertiary study conducted by Garousi and Mäntylä (2016). Thereafter, we conducted several searches on bibliographic databases to identify studies that focused on RT improvement in CI development environments.
On the basis of our analysis and results from the exploration phase, we identified a set of challenges and requirements (see the discussion of research gaps in Section 2.5).

We observe that the previously proposed techniques exhibit minor differences. (Elbaum et al., 2014; Hemmati et al., 2017; Kim & Porter, 2002; Marijan et al., 2013; Spieker et al., 2017; Srikanth et al., 2016; Strandberg et al., 2016). These techniques often share a common heuristic. Failure recurrence is the name of the heuristic applied, and it indicates that test cases that failed in previous builds have a much higher probability of failing again in the current build. Existing studies have suggested that HBTP can be effectively applied in CI environments. However, it has inherent limitations. For instance, not all regression faults can be captured effectively if there has been no previous failure, as with newly added test cases or those that have not previously revealed any failure. Thus, HBTP needs to be applied in combination with other TCP techniques.

A well-known heuristic in software testing is to maximize coverage as early as possible (Rothermel et al., 2001). To increase the likelihood of detecting faults, particularly when coverage information is unavailable, one can aim to maximize diversity as early as possible. The core idea is to spread the testing budget evenly across different parts of the system by performing a diverse set of test cases. This heuristic implies that test cases that capture the same faults tend to be more similar to each other, and test cases that capture different faults tend to be more dissimilar.

By taking both heuristics into account, one can perhaps detect regression faults more effectively than by applying them individually. HBD realizes this idea and combines the heuristics simultaneously. HBD is a blackbox, static, and minimal TCP technique. One might argue that whitebox TCP techniques are more effective than blackbox techniques, because the former have access to information about the SUT (e.g., code coverage). However, results from an experiment conducted by Henard et al. (2016) show a negligible difference between the effectiveness of whitebox and blackbox TCP techniques, that is, at most a 4% fault-detection-rate difference. This suggests that blackbox TCP techniques can be applied effectively where they are appropriate without a substantial compromise in the effectiveness of fault detection.

HBD is by definition a static technique, but it does not perform any exhaustive static analysis on the test cases, such as symbolic execution. The similarity measurement applied in HBD is based on lexicographic information associated with the test cases. Theoretically, it can be applied to all objects independent of their underlying structure. The other side of the coin is that this approach does not necessarily capture the semantics
behind the test cases. For instance, two test cases might be considered similar even though they are distant and correspond to different execution paths. This limitation was discussed by Ledru et al. (2012).

Given this limitation of HBD, one might argue that the coverage of “maximizing diversity as early as possible” cannot be as perfect as “maximizing coverage as early as possible”. This holds true. Yet, according to the study conducted by Henard et al. (2016), from the fault-detection perspective, the difference between these two strategies is minor. Furthermore, results from an experiment by Feldt et al. (2016) show that test sets with higher diversity (measured using NCD multisets) have higher structural and fault coverage in comparison to random selection. Thus, maximizing diversity as early as possible is a plausible strategy when coverage information is unavailable. In this way, we cast a wide net to detect a higher number of unique faults in the system.

HBD is a minimal TCP technique in the sense that the only information required is already encoded in the regression test suite, that is, information about individual test cases. Using such information, HBD can maximize diversity within the test suite as early as possible. During the later stages, when archived build logs are available, it can automatically analyze failure history and incorporate this knowledge during TCP. The raw build log, which is one of the outputs of CI builds, can be automatically collected from any CI server with the help of a simple script. One might assume that a large amount of historical data is required. However, our results suggest that HBD does not necessarily require a large amount of historical data and that its effectiveness improves to a certain degree with a larger history interval.

Implications: Taken together, the results from previous research phases have important academic and practical implications. From the academic perspective, results from the systematic mapping studies provide researchers with a synthesized and structured body of knowledge and a basis for future studies. The results from experiments provide empirical evidence in support of two previously proposed TCP heuristics—failure recurrence and test diversity—and their combination in CI development environments.

From the practitioner’s perspective, results from the systematic mapping studies provide a synthesized database and structured repository that may facilitate access to and comprehension of state-of-the-art knowledge. In addition, results from our experiments suggest that TCP in CI environments can be performed with negligible investment using build history and test distances. For the initial stages, when historical data are scarce or not yet available, one should spread the limited testing budget evenly across different parts of the system by running the most dissimilar test cases early in the process (i.e.,
maximizing diversity as early as possible). For the later stages, when archived build logs are available, one should give higher priority to the test cases that have higher historical value (i.e., those that revealed the most failure in the most recent revisions) and are most distant from those previously executed (i.e., cover different areas of the system).

5.2 Threats to validity

In the context of this dissertation, validity threats are classified into four distinct categories: construct validity, internal validity, external validity, and reliability (Wohlin et al., 2000). This section discusses the threats to the validity of the research along with the countermeasures taken during the research process.

5.2.1 Construct validity

Construct validity concerns the representativeness of the measures used to represent the concepts under investigation. The most important threat to the construct validity of systematic reviews relates to the identification of primary studies, that is, the search strategy and how well the data represent the phenomenon of interest. To mitigate these issues and reduce the risk of missing relevant studies, we followed an iterative approach and employed a search strategy that allowed us to identify as many primary studies as possible.

In Paper I, several known studies of CD were collected manually and closely investigated in order to better understand the phenomenon. Due to the undefined nature of the phenomenon under investigation, we developed a comprehensive search string with the aim of minimizing the risk of missing relevant studies. Our search string was based on three main attributes: deployment, continuity, and speed (and terms closely related to them). In Paper II, we performed several experimental searches and calculated the noise (the proportion of irrelevant studies to all retrieved records) and accuracy rate (the proportion of relevant studies to all retrieved records). This was possible using a candidate set of papers (87 studies of ASE) that were previously identified by conducting a search on Google Scholar. Given the broad scope of our subject, we performed several complementary searches with the aim of increasing the coverage of published studies.

Even though the objective of systematic reviews is to identify all relevant research on the subject of interest, it is more likely that such reviews capture only a sample of the relevant articles (Wohlin et al., 2013). One reason for this problem is that various groups
and research communities within the software engineering discipline use different
terms to refer to the same phenomenon. Thus, achieving 100% coverage of existing
publications is challenging. The objective of our search strategy was to retrieve as many
primary studies as possible within the scope of the research. Our search strategy had
an iterative design and was revised based on the several design decisions that were
thoroughly discussed in the original publications.

The selection of primary studies is another threat to the construct validity of literature
reviews. For instance, primary studies may be excluded as a result of the researcher’s
bias, subjective evaluation, or misjudgment of the study under evaluation. To mitigate
this threat, a thorough research protocol and a set of explicit inclusion and exclusion
criteria were established. To minimize the subjectivity of the selection process, each
step was conducted by a pair of researchers, and disagreements were resolved with the
help of a third reviewer. Publication bias is another threat to the construct validity of
literature reviews. This bias arises from “the problem that positive research outcomes
are more likely to be published than negative ones”. This problem particularly concerns
SLRs that aim to aggregate existing evidence and to evaluate and compare specific
methods or techniques. However, in SMSs, the effect of this threat is trivial, because
such analyses are usually not performed.

In the context of experiments (Papers IV and V), threats to construct validity concern
the use of proper measures and whether researchers are in fact measuring what they
intend to measure. To assess the effectiveness of TCP techniques, we applied the APFD
metric, which was originally introduced by Rothermel et al. (2001) and is the most
frequently used metric in the TCP literature (Khatibsyarbini et al., 2018). Apart from
effectiveness, we measured AMET in order to assess the performance of the techniques
under investigation.

5.2.2 Internal validity

Threats to internal validity concern the relationship between constructs and proposed
explanations. In the context of SMSs, internal validity is related to the researcher’s bias
in the extraction, analysis, and interpretation of data. To minimize the effect of this
threat, a data extraction form was developed. This form was piloted using a small set of
primary studies in order to establish a common understanding between the researchers.
The data extraction, analysis, and interpretation of primary studies were performed by
pairs of researchers at all stages, and disagreements were further resolved with the help
of a third researcher and collective discussion. This enabled triangulation in all research phases and minimized the possibility of researcher bias.

In the context of experimental studies, threats to internal validity are associated with potential faults in implementation (e.g., TCP algorithms or the experiment itself). To minimize the likelihood of errors in our implementation, we took several countermeasures into consideration. For instance, to validate the data gathered from build logs, we randomly selected several build logs from each project and manually verified the extracted data. Our implementation was piloted on a small sample before running the actual experiment. Furthermore, the implementation and results were discussed and reviewed by the coauthors in regular meetings. Overall, our implementation underwent several iterations and validations before the actual experiment was conducted.

5.2.3 External validity

This validity threat concerns the potential generalizability of the results outside the subject area in which they were originally investigated. In Papers I and II, external validity is closely related to our search strategy and the question of whether the data represent the investigated phenomenon. Paper I had a narrow scope and included all existing studies on the subject area published between 2001 and 2014. Paper II had a broad scope but included a large and representative sample of relevant studies. This review included 679 studies from multiple research subareas, published between 1990 and 2015. These studies were published in over 289 unique publication venues and include author affiliations with 398 unique institutions in 35 countries. These figures may indicate that our primary studies are not limited to or biased towards a specific time period, publication venue, research institute, or geographical area.

In the context of experimental studies (Papers IV and V), threats to external validity concern whether the subjects of our studies are representative of real programs. In Paper IV, we investigated six open-source software projects, each of which included several builds over a long period of time (a total of 1,330 faulty builds between 2013 and 2016). In Paper V, we investigated another six open-source software projects that were extracted from the Defect4j dataset (a total of 395 faulty builds between 2006 and 2015). Thus, the subjects of our experiments are real-world software projects and include real-world CI builds, test suites, and regression faults.
5.2.4 Reliability

Threats to reliability concern the repeatability of the research and the possibility of coming to the same conclusion reached by the original study. In context of SMSs, the search process and retrieval of the studies are part of an objective process that must be repeatable by strictly implementing the same research protocol. However, the primary study selection and the data extraction, interpretation, and classification of the literature were subjective and belong to the categories of research creativity and research contribution.

In the context of experimental studies, repeatability required access to the data that were used and a thorough report on the research process that was applied. The study design, along with a careful explanation of our implementation, was reported in the original publications. Furthermore, the data we used in both of our experiments, that is, TravisTorrent (Beller et al., 2017) and Defect4j (Just et al., 2014), are publicly available.

5.3 Limitations

The problem formulation and solution development in this dissertation were based on the literature analysis and on the author’s limited experience in the software industry. The data collection method was archival research, and the experiments were conducted in a laboratory setting rather than in an industrial context. Consequently, our conclusions are based on the literature analysis and the ex post analysis of software artifacts. Therefore, limited connection with industry is the 1st and main limitation of the dissertation. Needless to say, design decisions during the research process (see Section 3.1) were made on the basis of the research problem, and the author faced several constraints.

The 2nd limitation of this dissertation is related to the metrics used in our experiments. The APFD metric used in our experiments was concerned with the number of test cases executed rather than the actual RT time. To perform a comprehensive assessment, one needs to measure the end-to-end time, that is, the TCP execution time, in addition to the actual test suite execution time. Such measurement requires downloading, compiling, and running the test suite for all the revisions examined. This procedure involves manual effort and was not possible due to the large number of faulty revisions analyzed in our experiments (1,330 in Paper IV and 395 in Paper V). The automation of the entire procedure was technically possible, but it was infeasible given the scope and constraints of our studies.
The 3rd limitation—closely related to the 1st—is the lack of a research framework with which to guide the design and evaluation of the solution in a systematic and iterative manner. To address this limitation, design science research (Hevner, March, Jinsoo, & Ram, 2004) could have been applied. The design science research methodology (Hevner et al., 2004; Peffers, Tuunanen, Rothenberger, & Chatterjee, 2007) includes several research cycles and concrete procedures that guide researchers in the effective development and evaluation of industry-oriented solutions. Design science research requires close collaboration with industry and planning during the early stages of the research process.

The 4th limitation is related to the design of the HBD technique. Failure recurrence, which is one of the heuristics we applied, assumes that test cases have deterministic outcomes. However, this assumption does not always hold. There are test cases known as “flaky tests” that, for one reason or another, have nondeterministic outcomes. The practice commonly used to deal with a flaky test is the rerun technique, that is, running the failing test multiple times to determine its verdict (Luo, Hariri, Eloussi, & Marinov, 2014). The rerun technique is supported by several testing frameworks and is enabled by the use of annotations in Java. For instance, Android has @FlakyTest ⁹, Spring has @Repeat ¹⁰, and JUnit has an extension ¹¹ for labeling a test that requires reruns upon failure. To minimize its impact, one needs to distinguish whether a failure is due to a flaky test or a recently introduced regression. This can be done by processing annotations or applying an automated approach such as DeFlaker, which was recently proposed by Bell et al. (2018). Once flaky tests have been identified, one needs to devise strategies for minimizing their impact when performing TCS and TCP using historical data. To the best of our knowledge, these issues have not been widely investigated, particularly in the context of RT improvement techniques.

5.4 Future work

From an academic perspective, the results of this dissertation indicate several opportunities for further research. One such opportunity is the implementation of HBD in the software industry and its evaluation with respect to several variables. The utility of HBD in our experiments was evaluated using two variables: effectiveness (i.e., average

⁹https://developer.android.com/reference/android/test/FlakyTest
¹⁰https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/test/annotation/Repeat.html
¹¹https://github.com/artsok/rerunner-jupiter/
percentage of faults detected) and performance (i.e., method execution time). However, other variables, such as efficiency (i.e., the extent to which the technique reduces the overall RT time), scalability (i.e., the ability to work with large regression test suites), and generality (i.e., the ability to operate in a wide range of situations), should be considered. Once HBD is integrated with the CI server, measurements with respect to several variables can be automatically recorded. Furthermore, qualitative data can be gathered from practitioners to investigate the generality of HBD and identify potential areas for its improvement.

Using qualitative data from practitioners, one can also identify and incorporate the domain expert knowledge and other heuristics into TCP. HBD has a two-stage prioritization mechanism, that is, scoring and clustering test cases on the basis of their historical value (i.e., those that revealed the most failure in the most recent revisions) and then iteratively reprioritizing the intracluster tests on the basis of their distance to the set of already prioritized tests. The 1st stage can be easily expanded to incorporate other heuristics, e.g., newly added test cases should have a higher priority, because they may correspond to the recent changes in the system. In this way, one can calculate priorities for the test cases from several perspectives, combine them into a single value, and cluster test cases accordingly. The 2nd stage aims to maximize diversity as early as possible and can be performed regardless of what was done in the previous stage.

To maximize diversity as early as possible, one needs to measure the similarities among the test cases and then leverage this information to prioritize them. The similarity measurement in our experiments was performed using lexicographic information associated with the test cases. However, this approach does not necessarily capture the semantics behind the test cases, which may lower the precision of measurement. Future studies are required to investigate approaches that can capture more information about test cases (e.g., the syntactic structure or semantics of a program) and perform similarity measurement with higher precision. This improvement in precision should not come with a high overhead; otherwise, its application will remain merely theoretical.

Once similarity measurement has been performed, this information should be exploited to perform prioritization. One could argue that diversification is the best strategy when no strong clues about fault-revealing test cases are available. The opposite viewpoint favors the intensification strategy, according to which the testing budget is consumed by and around the most probable fault-revealing test cases. Theoretically, both strategies can be applied simultaneously (i.e., intensification where necessary and diversification of the remaining budget). However, making decisions about when
and how to apply these strategies, either individually or in combination, remains a challenge. To the best of our knowledge, neither the application of these strategies nor their relevance and impact have been widely investigated by researchers. The only exception we are aware of is the recent study by Patrick and Jia (2017), which investigated the trade-off between the two exploitation strategies in adaptive random testing.

Regardless of which exploitation strategy is chosen, a TCP algorithm needs to iteratively find the item that is most (dis)similar to the set of already prioritized test cases. This can be realized using a pairwise algorithm. However, it comes with the cost of pairwise comparison, and its performance becomes inefficient as the test suite becomes larger. The underlying issue can be defined as a similarity search problem, which involves searching within a large set of objects for a subset of objects that closely resemble a given query object. In this dissertation, we have investigated LSH, which was originally introduced by Indyk and Motwani (1998) and recently proposed by Miranda, Cruciani, Verdeccchia, and Bertolino (2018) to be applied in TCP. However, several other approaches have been proposed for the similarity search problem. To effectively apply them to software engineering problems, the existing body of knowledge must be explored systematically and potential approaches need to be investigated and adapted.

To achieve continuous RT, one needs to implement several RT techniques. These techniques can be applied in combination and in a complementary fashion, using common input resources available in any CI development environment. TCS can be applied during the presubmission phase, when changes have not yet been integrated with the repository and immediate feedback is required. TCS can be effectively performed using build history (e.g., Ekelund and Engstrom (2015); Elbaum et al. (2014); Knauss et al. (2015); Spieker et al. (2017)). TCP should be performed independently and during postsubmission testing. TCP can also be applied in combination with TCS and during presubmission testing. In this way, we not only run the subset of test cases affected by the recent changes but do so in an order that enables the earlier detection of regression faults. TSM should also be applied frequently. This can be achieved even without code-coverage criteria and by using the distances among the test cases (Cruciani et al., 2019; Feldt et al., 2016). Future studies are required to integrate these techniques into a comprehensive solution that can be easily adopted in the software industry.
6 Conclusions

RT in combination with CI is considered a good software development practice, because it ensures that recent changes do not introduce any regression into the system. However, the combination creates several challenges for the software industry. The execution of a large test suite at each build cycle is not realistic. This is simply because the testing budget is often limited in industry, and RT needs to make the most of limited resources. To detect regression faults earlier, one alternative is to permute the test cases within the test suite.

This dissertation aims to shorten the build cycle in CI environments by detecting regression faults earlier, allowing software developers to integrate and verify their changes more frequently and continuously. The research was performed in three phases. In the 1st phase, the relevant literature was systematically surveyed to analyze the existing body of knowledge and identify the potential research opportunities. In the 2nd phase, the TCP technique of HBD was developed and underwent several rounds of improvements and evaluations. The technique has a two-stage prioritization mechanism, that is, scoring and clustering test cases on the basis of their historical value and then iteratively reprioritizing the intracluster tests on the basis of their distance to the set of already prioritized tests. In the 3rd phase, results from the previous two phases were synthesized in order to draw conclusions and outline implications.

Our results bring to mind the well-known adage “don’t put all your eggs in one basket”. For the initial stages, when no information about the SUT is available, one should spread the limited testing budget evenly across different parts of the system by running the most dissimilar test cases early in the process (i.e., maximizing diversity as early as possible). For the later stages, when historical build data are available, one should give higher priority to the test cases that have higher historical value (i.e., those that revealed the most failure in the most recent revisions) and are most distant from those previously prioritized (i.e., cover different areas of the system).

A major contribution of this dissertation is the design, implementation, and experimental evaluation of a TCP technique that can be applied in CI environments. Our experiments provide empirical evidence in support of two previously proposed TCP heuristics (i.e., failure recurrence and test diversity) and their combination in CI development environments. Furthermore, results from the systematic mapping
studies contribute to the literature by providing a synthesized and structured body of knowledge and a basis for future studies. We conclude that TCP can be performed in CI environments with negligible investment using build history and test distances.
References


Caliebe, P., Herpel, T., & German, R. (2012). Dependency-based test case selection and prioritization in embedded systems. In Software testing, verification and validation (icst), 2012 ieee fifth international conference on (pp. 731–735).

of the 29th ACM/IEEE International Conference on Automated Software Engineering (pp. 55–66).


doi: 10.1109/ICSE.2019.00055


prioritization approaches in regression testing: A systematic literature review. Information and Software Technology, 93, 74–93.


Wohlin, C., & Aurum, A. (2015). Towards a decision-making structure for select-


Appendix 1 Technical details

1.1 Similarity measurement

Several similarity metrics have been proposed in the literature, and they have been used in many areas, such as comparative genomics, music classification, and plagiarism detection. In our experiments, we applied five different approaches, which were selected from the literature based on the results of experimental studies (Feldt et al., 2016; Feldt, Torkar, Gorschek, & Afzal, 2008; Hemmati & Briand, 2010; Ledru et al., 2012; Miranda et al., 2018).

The 1st metric applied is the Manhattan distance. The Manhattan distance between two objects is the sum of the differences of their corresponding components. In our implementation, similar to Ledru et al. (2012), we converted source code to a vector of numbers. A string of size \( n \) can be seen as a vector of characters in an \( n \)-dimensional space, and characters can be associated with their ASCII code (or any other numerical coding system). The Manhattan distance is presented as

\[
\sum_{i=1}^{n} |x_i - y_i|,
\]

where \( x \) and \( y \) are the strings to be compared.

The 2nd metric applied is the Jaccard distance. The Jaccard similarity between two sets \( x \) and \( y \) is defined as

\[
JS(x, y) = \frac{|x \cap y|}{|x \cup y|},
\]

and their distance is defined as

\[
JD(x, y) = 1 - JS(x, y).
\]

To calculate the Jaccard distance, the source code is converted to a set of k-shingles, that is, any substring of length \( k \) found within the text. In our experiments, we used 5-shingles, which is commonly used in the analysis of relatively short documents (Leskovec, Rajaraman, & Ullman, 2014).

The 3rd metric applied is called normalized compression distance (NCD) (Cilibrasi & Vitányi, 2005; Li, Chen, Li, Ma, & Vitányi, 2004). Two objects are deemed similar if we can significantly compress one given the information in the other. NCD is given as follows:

\[
NCD(x, y) = \frac{C(xy) - \min\{C(x), C(y)\}}{\max\{C(x), C(y)\}},
\]

where \( x \) and \( y \) are the strings to be compared. NCD relies on a compressor function \( C \) that calculates the approximate Kolmogorov complexity and returns the length of the input string after its compression, using a chosen compression program. In our experiments, we used LZ4, which is a high-speed lossless data compression algorithm.\(^{12}\)

---

\(^{12}\)The LZ4 compression algorithm and its implementation is available at http://lz4.github.io/lz4/
The 4th metric applied is NCD multisets, which was recently proposed by Cohen and Vitányi (2015). The difference between NCD and NCD multisets is that the latter perform similarity measurements at the level of entire sets of elements rather than between pairs.

The 5th approach applied is locality-sensitive hashing (LSH), which was originally proposed by Indyk and Motwani (1998). LSH is not a metric per se but a popular solution to the similarity search problem. LSH hashes input items so that similar items map to the same buckets with high probability.

In our experiments, we implemented LSH using the MinHash technique to rapidly estimate the Jaccard similarity. In our implementation, we followed the instructions provided by Leskovec et al. (2014), which are also described here. To estimate the Jaccard similarity, we converted the source code to a set of 5-shingles. However, their size is often large, and it is impractical to use them directly.

Using the MinHash technique, we replaced these sets with a much smaller representation (e.g., a signature) while preserving the Jaccard similarity between them. Given a hash function \( h \) and an input set \( S \), we hashed all elements in the set using the hash function and picked the minimum resulting value as MinHash of \( S \). This process was repeated \( P \) times (i.e., the number of permutations) using different hash functions to calculate the signature of a set (e.g., a sequence of MinHashes). Thereafter, the Jaccard similarity of two sets can be estimated using the fraction of common MinHashes in their signature. However, similarity searches among large numbers of pairs are still inefficient.

LSH works with a signature matrix (e.g., MinHash signatures as column) and divides it into \( b \) bands consisting of \( r \) rows each. For each band, LSH takes vectors of numbers (e.g., the portion of one column within that band) and hashes them to the buckets using a hash function. The more similar two columns are, the greater their likelihood of colliding into some bands. When two items fall into the same bucket, it means a portion of their signature agrees, and they will be added to the candidate set. The candidate set returned by an LSH query contains only a subset of items that have a greater likelihood of being similar (e.g., that have a Jaccard similarity over a certain threshold). An approximation of this threshold is defined as \( ST = (1/b)(1/r) \).
1.2 TCP algorithms

To maximize diversity as early as possible, one needs to measure the distances among the test cases and then leverage this information to perform TCP. To calculate the distance between a test case \( t \) and set of test cases \( T' \) using the metric \( d \), Ledru et al. (2012) proposed the following generic distance function:

\[
distance(t, T', d) = \min \{d(t, t_i) | t_i \in T' \text{ and } t_i \neq t\}.
\]

The min operation was used in the above function, because an empirical study by Jiang et al. (2009) showed that maximize-minimum is more efficient than maximize-average and maximize-maximum. Ledru et al. (2012) also proposed a pairwise algorithm (Algorithm 1) that iteratively picks the most dissimilar test case, that is, the one with the greatest distance from a set of already prioritized test cases. In our experiments, we implemented the pairwise algorithm using different similarity metrics, that is, Manhattan, Jaccard, and NCD.

**Algorithm 1:** Maximize diversity using pairwise algorithm.

**Data:** Test Suite \( TS \) and Distance Metric \( d \)

**Result:** Prioritized Suite \( PS \)

1. Find \( t \in TS \) with the maximum \( distance(t, TS, d) \);
2. Append \( t \) to \( PS \) and remove from \( TS \);
3. **while** \( TS \) is not empty **do**
   4. Find \( t \in TS \) with the maximum \( distance(t, PS, d) \);
   5. Append \( t \) to \( PS \) and remove from \( TS \);
4. **end**

For the NCD multisets, we adapted the pairwise algorithm (Algorithm 2) so that at each iteration, we pick a test \( t \in TS \) that has maximum Kolmogorov complexity when compressed with the entire set of the already prioritized suite \( PS \). This means that the candidate test has less mutual information with \( PS \) and is more different than any other \( t \in TS \). Note that \( C \) is a function that calculates the approximate Kolmogorov complexity and returns the length of the input string after its compression, using a chosen compression program.

To employ LSH for the purpose of TCP, we implemented an algorithm (Algorithm 3) proposed by Miranda et al. (2018). LSH is often configured with a high similarity
Algorithm 2: Maximize diversity using NCD multisets.

**Data:** Test Suite $TS$

**Result:** Prioritized Suite $PS$

1. while $TS$ is not empty do
   2. Find $t \in TS$ with the maximum $C(PS, t)$;
   3. Append $t$ to $PS$ and remove from $TS$;
4. end

threshold so that the candidate set contains only highly similar items. However, in our context, we are interested in items with a maximum distance from the LSH query. Thus, similar to Miranda et al. (2018), we configured LSH in order to achieve an approximately 0.1 similarity threshold. In this way, the candidate set $CS$ would contain almost all test cases, and the distant set $DS$ would include a small number of remaining items with high Jaccard distance.

Algorithm 3: Maximize diversity using locality-sensitive hashing.

**Data:** Test Suite $TS$

**Result:** Prioritized Suite $PS$

1. $signatures \leftarrow \text{MinHashSignature}(TS)$;
2. $\text{LSH}.\text{Index}(signatures)$;
3. $query \leftarrow \text{MinHashSignature}(\emptyset)$;
4. while $signatures$ is not empty do
   5. $CS \leftarrow \text{LSH}.\text{Search}(query)$;
   6. $DS \leftarrow signatures - CS - PS$;
   7. Find $i \in DS$ with the maximum distance to $PS$;
   8. Append $i$ to $PS$ and remove from $signatures$;
   9. $query \leftarrow \text{Update cumulative MinHash signature of } PS$;
10. end

The last algorithm applied in our experiments is HBD (Algorithm 4). This algorithm has a two-stage prioritization mechanism. During the 1st stage, we calculated the cumulative priority for each test using its previous failures over the last $N$ builds (depending on the interval size). The highest weight corresponds to the failure exposed

---

13LSH parameters: permutations: 10; bands: 10; rows: 1
in a previous build (current−1), and the failure in every preceding build is weighted lower than the failure in its successive build. Specifically, failures are weighted by their distance \( W_n \), which reflects the impact of the failure that occurred in the past and is \( n \) builds distant from the current build session. For each \( W_n \), a value between 0.9 (\( n = 1 \)) and 0.1 (\( n \geq 9 \)) was assigned. Thereafter, we aggregated tests into clusters based on their historical value and sorted these clusters in descending order. During the 2\(^{nd} \) stage, we iteratively reprioritized the intracluster tests on the basis of their distance to the set of already prioritized tests. This can be realized using the similarity-based TCP algorithms presented above.

Algorithm 4: TCP using history-based diversity.

**Data:** Test Suite \( TS \) and Build History \( buildHistoryDatabase \)

**Result:** Prioritized Suite \( PS \)

/* First stage prioritization */

1. \( clusters \leftarrow \text{new AssociativeArray}; \)

2. \( \text{foreach test } t_i \in TS \) do

3. \( \quad \text{verdicts} \leftarrow buildHistoryDatabase.find(t_i); \)

4. \( \quad \text{score} \leftarrow \text{calculateHistoricalValue}(\text{verdicts}); \)

5. \( \quad \text{if clusters.contains(score) then} \)

6. \( \quad \quad \text{clusters.get(score).add(t_i);} \)

7. \( \quad \text{else} \)

8. \( \quad \quad \text{items} \leftarrow \text{new List}; \)

9. \( \quad \quad \text{items.add(t_i);} \)

10. \( \quad \quad \text{clusters.put(score,items);} \)

11. \( \text{end} \)

12. \( \text{end} \)

13. \( \text{clusters.sortDescendingByKey();} \)

/* Second stage prioritization */

14. \( \text{foreach cluster } c_i \in clusters \) do

15. \( \quad \text{intraClusterTests} \leftarrow c_i.values; \)

16. \( \quad \text{ordered} \leftarrow \text{maximizeDiversity}(\text{intraClusterTests},PS); \)

17. \( \quad PS.addAll(\text{ordered}); \)

18. \( \text{end} \)
Original publications


Reprinted with permission from Elsevier (I, II, and IV), The Institute of Electrical and Electronics Engineers, IEEE (III) and Springer Nature (V).

Original publications are not included in the electronic version of the dissertation.
727. Hurskainen, Sonja (2018) The roles of individual demographic history and environmental conditions in the performance and conservation of northern orchids
729. Tokkonen, Helena (2019) Say, Do, Make! : user involvement in information systems design
731. Mylonopoulou, Vasiliki (2019) MAD : designing social comparison features in health behaviour change technological interventions
732. Shevchuk, Nataliya (2019) Application of persuasive systems design for adopting green information systems and technologies
733. Tripathi, Nirnaya (2019) Initial minimum viable product development in software startups : A startup ecosystem perspective
735. Salman, Ilfrac (2019) The effects of confirmation bias and time pressure in software testing
736. Hosseini, Seyyedrevar (2019) Data selection for cross-project defect prediction
737. Karvonen, Juhani (2019) Demography and dynamics of a partial migrant close to the northern range margin
738. Rohunen, Anna (2019) Advancing information privacy concerns evaluation in personal data intensive services
Alireza Haghighatkhah

TEST CASE PRIORITIZATION USING BUILD HISTORY AND TEST DISTANCES

AN APPROACH FOR IMPROVING AUTOMOTIVE REGRESSION TESTING IN CONTINUOUS INTEGRATION ENVIRONMENTS