Linköping Studies in Science and Technology
Dissertation No. 2247

Contributions to Improving Feedback and Trust in Automated Testing and Continuous Integration and Delivery

Azeem Ahmad

LINKÖPING UNIVERSITY

Linköping Studies in Science and Technology
Dissertations, No. 2247

Contributions to Improving Feedback and Trust in Automated Testing and Continuous Integration and Delivery

Azeem Ahmad

Linköping University
Department of Computer and Information Science
Software and Systems
SE-581 83 Linköping, Sweden

Linköping 2022

Edition 1:1

ISBN 978-91-7929-422-9 (print)
ISBN 978-91-7929-423-6 (PDF)
ISSN 0345-7524
DOI https://doi.org/10.3384/9789179294236

Published articles have been reprinted with permission from the respective copyright holder.
Typeset using LaTeX

Printed by LiU-Tryck, Linköping 2022

What is a man's worth, here today, and gone tomorrow.

ABSTRACT

An integrated release version (also known as a release candidate in software engineering) is produced by merging, building, and testing code on a regular basis as part of the Continuous Integration and Continuous Delivery (CI/CD) practices. Several benefits, including improved software quality and shorter release cycles, have been claimed for CI/CD. On the other hand, recent research has uncovered a plethora of problems and bad practices related to CI/CD adoption, necessitating some optimization. Some of the problems addressed in this work include the ability to respond to practitioners' questions and obtain quick and trustworthy feedback in CI/CD. To be more specific, our effort concentrated on: 1) identifying the information needs of software practitioners engaged in CI/CD; 2) adopting test optimization approaches to obtain faster feedback that are realistic for use in CI/CD environments without introducing excessive technical requirements; 3) identifying perceived causes and automated root cause analysis of test flakiness, thereby providing developers with guidance on how to resolve test flakiness; and 4) identifying challenges in addressing information needs, providing faster and more trustworthy feedback.

The findings of the research reported in this thesis are based on data from three single-case studies and three multiple-case studies. The research uses quantitative and qualitative data collected via interviews, site visits, and workshops. To perform our analyses, we used data from firms producing embedded software as well as open-source repositories. The following are major research and practical contributions.

- **Information Needs:** The initial contribution to research is a list of information needs in CI/CD. This list contains 27 frequently asked questions on continuous integration and continuous delivery by software practitioners. The identified information needs have been classified as related to *testing*, *code & commit*, *confidence*, *bug*, and *artifacts*. We investigated how companies deal with information needs, what tools they use to deal with them, and who is interested in them. We concluded that there is a discrepancy between the identified needs and the techniques employed to meet them. Since some information needs cannot be met by current tools, manual inspections are required, which adds time to the process. Information about code & commit, confidence level, and testing is the most frequently sought for and most important information.
- **Evaluation of Diversity Based Techniques/Tool:** The contribution is to conduct a detailed examination of diversity-based techniques using industry test cases to determine if there is a difference between diversity functions in selecting integration-level automated test. Additionally, how diversity-based testing compares to other optimization techniques used in industry in terms of fault detection rates, feature coverage, and execution time. This enables us to observe how coverage changes when we run fewer test cases. We concluded that some of the techniques can eliminate up to 85% of test cases (provided by the case company) while still covering all distinct features/requirements. The techniques are developed and made available as an open-source tool for further research and application.
- **Test Flakiness Detection, Prediction & Automated Root Cause Analysis:** We identified 19 factors that professionals perceive affect test flakiness. These perceived factors are divided into four categories: test code, system under test, CI/test infrastructure, and organizational. We concluded that some of the perceived factors of test flakiness in closed-source development are directly related to non-determinism, whereas other perceived factors concern different aspects e.g., lack of good properties of a test case (i.e., small, simple and robust), deviations from the established

processes, etc. To see if the developers' perceptions were in line with what they had labelled as flaky or not, we examined the test artifacts that were readily available. We verified that two of the identified perceived factors (i.e., test case size and simplicity) are indeed indicative of test flakiness. Furthermore, we proposed a light weight technique named *trace-back coverage* to detect flaky tests. *Trace-back coverage* was combined with other factors such as test smells indicating test flakiness, flakiness frequency and test case size to investigate the effect on revealing test flakiness. When all factors are taken into consideration, the precision of flaky test detection is increased from 57% (using single factor) to 86% (combination of different factors).

POPULÄRVETENSKAPLIG SAMMANFATTNING

Många programvaruutvecklande organisationer har under de senaste åren börjat arbeta i stor skala med kontinuerlig integration och leverans (på engelska förkortat CI/CD). Det innebär att programkoden regelbundet och ofta integreras, byggs och testas – till stor del automatiskt med hjälp av en svit med stödverktyg. Resultatet av CI/CD-processen är en ny version av programvaran som teoretiskt sett skulle kunna levereras till kund varje dag, men det tillkommer praktiska och organisatoriska moment som inte alltid kan genomföras så ofta. Införande av CI/CD ger många fördelar, inklusive högre kundnöjdhet, frekventa programvaruutgåvor, förbättrad produktkvalitet och utvecklareffektivitet samt tidig identifiering av fel. CI/CD erbjuder å andra sidan många nya utmaningar för utvecklare och forskare.

Komplexiteten hos CI/CD till följd av dess omfattning är en av utmaningarna. CI/CD stöder ett stort antal intressenter som projektledare, utvecklare, testare, produktägare med flera och omfattar miljontals programvaruartefakter (källkod, felrapporter, byggfiler, testfall osv.). Intressenterna behöver i sin vardag leta efter information som genereras av CI/CD-processen. Att leta efter svaren på frågor som "Hur stort förtroende har vi för en specifik testsvit?" eller "Är vi redo att släppa en specifik version av programvaran?" har visat sig vara svårt eftersom man för att generera svaren behöver integrera information från en kombination av artefakter. Den höga frekvensen med vilken programkod revideras, byggs, testas och distribueras resulterar i en ökning av det totala antalet artefakter, vilket också gör det svårare att svara på sådana frågor. Till detta kommer utmaningen att dessa frågor kan tolkas på olika sätt, vilket gör det ännu svårare att svara på dem korrekt. Intressenter som spenderar mycket tid och möda för att leta efter sådan information kan bli distraherade från produktivt arbete.

För att hantera denna utmaning har vi genomfört en större intervjustudie som resulterat i en lista över informationsbehov från intressenter på olika företag bestående av 27 vanliga frågor som för att besvaras behöver information från CI/CD-processen. Informationsbehoven har delats in i fyra klasser och vi har samlat data om deras betydelse, förväntad frekvens och kostnad för att ta itu med dem. Vi har också information om vilka verktyg företagen använder och vi har beskrivit ett system för att visualisera händelser i CI/CD-processerna som teoretiskt kan svara mot alla informationsbehov. En forskningsprototyp som använder det så kallade Eiffel-protokollet finns tillgänglig som öppen källkod.

En annan utmaning som introduceras av CI/CD är den hastighet med vilken återkoppling tillhandahålls under regressionstestning eftersom antalet körda tester ökar avsevärt på grund av frekventa förändringar i systemet under test och en ständig tillväxt av antalet testfall allteftersom fler funktioner implementeras.

För att möta denna utmaning genomförde vi en detaljerad undersökning av metoden att välja ut ett mindre antal testfall för snabbare återkoppling under dagtid. Principen för urvalet kallas diversitetsbaserad testning och innebär att vi automatiskt väljer testfall som är så olika som möjligt med antagandet att de också representerar testning av olika egenskaper hos systemet som testas. I en fallstudie kunde vi observera att det gick att utesluta 85% av testfallen samtidigt som alla viktiga egenskaper hos systemet blev testade med åtminstone ett testfall var. Återkopplingscykeln i fallstudien kunde därmed reduceras från 3 timmar till 30 minuter. Företaget i fallstudien kompletterar detta med att köra samtliga testfall under natten då snabb återkoppling inte behövs. Teknikerna har implementerats och gjorts tillgängliga som ett verktyg med öppen källkod för vidare forskning och tillämpning.

Den tredje utmaningen är relaterad till förtroendet för resultatet från CI/CD-processen. Snabbare återkoppling från CI/CD-testcykler är ett framsteg, men kvalitén på den feedback som tillhandahålls och utvecklarnas förtroende för den är lika viktiga. Utvecklarens produktivitet blir lidande när testresultaten är icke-deterministiska, dvs. att man inte kan lita på om det som testningen indikerar som fel är verkliga brister. Sådan instabila (engelska: flaky) tester växlar mellan att signalera att testningen lyckades och att den misslyckades trots att utvecklaren inte gjort några ändringar i programkoden som testas.

Många storskaliga programvaruprojekt uppvisar en problematisk mängd instabila tester. För att möta denna utmaning identifierade vi genom arbetsmöten och enkäter 19 faktorer som professionella utvecklare och testare uppfattar påverkar instabilitet hos tester. Dessa upplevda faktorer är indelade i fyra kategorier: testfallskod, det testade systemet, infrastruktur och företagens organisation. Vi drog slutsatsen att några av de upplevda orsakerna till instabilitet i testningen hos företag är direkt relaterade till egenskaper hos det testade systemet medan andra involverar olika aspekter, såsom brist på lämpliga testfallsegenskaper, avvikelser från etablerade processer och ad hoc-beslut. Vi undersökte också ett antal testfall som utvecklarna angivit som instabila och kunde verifiera att upplevda faktorer, bland annat att stora testfall och testfall med hög komplexitet, har större benägenhet att uppvisa ett instabilt beteende.

Baserat på detta har vi utvecklat och utvärderat ett verktyg för att identifiera instabila tester i projekt med öppen källkod. Verktyget använder maskininlärning för att väga samman en rad olika möjliga orsaker till instabilitet och vi kunde observera att noggrannheten i verktygets prediktion ökade ju fler faktorer som togs med. Data om de olika faktorerna är relativt enkla att ta fram på automatisk väg ur CI/CD-processen.

Acknowledgments

Even though this dissertation has been reviewed more than once, it has a magical ability to hide special characters in places where they don't belong. For instance, this dissertation might all of a sudden show you Table? or [?]. The dissertation is the only one to blame. I read somewhere that there is a woman behind every successful man. If you want to call this dissertation a success, I would like to thank not just one but three women: Saleem Naz (mother), Maleeha Azeem (wife), and Khola Ahmad (daughter). Without them, this dissertation would not have been finished. I believe I may amend the proverb because Ibrahim Ahmad, my 5-year-old son, has also served as an inspiration to me through his inquisitive questions, which have piqued my own curiosity. My mother, who would never be able to read it because she never went to school and could not read, urged me to pursue a PhD. Because of their age, my daughter and son are also unable to read. But my wife will undoubtedly read it and conclude that this dissertation would have finished one year early, if she had allowed me to work on weekends or public holidays.

Kristian Sandahl and Ola Leifler have not only been a torchbearer for me, but for everyone who has worked with them. Their capacity to initiate critical, serious, instructive, and engaging conversations is unmatched. It was a privilege to work with them and to learn from them. In addition to providing me technical knowledge, they have also shown me how to cultivate veggies if necessary. I appreciate both of you. Aseel Berglund's commitment to gamification has motivated me. You taught me a lot, thank you. I appreciate you as well.

Now comes the difficult part, and read it at your own risk because there are so many names in the following text that deserve to be thanked but may bore the audience. Francisco Gomes de Oliveira Neto (this is one person) who has been a huge assistance to me and the nicest person I have ever met. He has not only inspired me with his own work, but he has also encouraged me with Dos and Don'ts in the PhD. In particular, I'd want to express my gratitude for the support I received from people/things such as Eduard Enoiu, Anne Moe, Lene Rosell, Martin Sjölund, Ola Söder at Axis Communication, the Massage Chair in corridor, and PELAB at various points throughout my PhD. Together with my gratitude, I also feel guilty for Nahid Shahmehri, who worked so hard to convinced me to play badminton. I'm certain I've offered her a number of excuses she hasn't heard before. It is important to acknowledge one individual who shaped my educational path when I decided to abandon education in favor of cheap labor. Shaukat Ali Shaukat, a teacher at Govt. Saleem Model High School, came to my residence to persuade my family of the value of education and to demonstrate his unwavering confidence in me. Special thanks to those friends who began referring to me as Dr. during my master's studies even though I lacked a PhD and put pressure on me to pursue a PhD.

This work was supported by Linköping University and Software Center: project 18 (Data Visualization in CI/CD) and project 30 (Aspects of Automated Testing).

Azeem Ahmad, Linköping, August 2022

Contents

1 Introduction

Software engineering techniques and concepts have evolved dramatically over the years, resulting in better quality software. As part of the Continuous Integration and Continuous Delivery (CI/CD) approach, code is regularly merged, built and tested in order to produce an integrated version (i.e., release version) that can be shared among the team members. The CI/CD processes are built into a pipeline, which has been referred to by several terms such as '(continuous) integration pipeline' [1], [2], 'deployment pipeline' [1], 'continuous delivery pipeline' [3], [4], and 'continuous integration and delivery pipeline' [5]. For this dissertation, we adopt Humble and Farley's [1] definition of continuous integration and delivery: "the practice of all developers often committing their changes, and that each change is considered a release candidate to be validated by the build and test process." The detailed context of this research (i.e., scope of research) is provided in Section 1.1.

The adoption of CI/CD provides many benefits such as increased customer satisfaction [6], frequent software releases [6]–[8], improved product quality and developer's productivity [6]–[9], and early bug detection [8]. On the other hand, CI/CD introduces numerous challenges. One of the challenges is the *complexity* of CI/CD as a result of its scale. For instance, CI/CD supports many stakholders (i.e., project managers, developers, testers, product owners, etc.), teams (e.g., feature, maintenance, deployment, etc.), and thousands of software artifacts (source code, bug reports, backlog items, etc.) [10]. Stakeholders seek out the information generated/embedded by/in the software artifacts in CI/CD pipeline [11]–[14]. Looking for answers to questions such as *"How much confidence do we have in a specific test suite?"* or *"Are we ready to release a specific version of the software?"* have proven difficult because the answers integrate information from a combination of different kinds of artifacts. The high frequency with which changes are committed, built, tested, and deployed results in an increase in the total number of artifacts, making it more difficult to answer such questions. The fact that these questions can be interpreted differently makes it more challenging to answer them correctly. Developers that spend a considerable amount of time and effort to identify such information can be distracted from doing productive work.

Identifying these information needs can help us better understand the tools, practices, and processes that are important when addressing those information needs [11], [12]. A better understanding of the information needs of software practitioners has several benefits, such as staying competitive, increasing awareness of the issues that can hinder a timely release, and building a visualization tool that can help practitioners to address their information needs.

Another challenge introduced by CI/CD is the speed with which feedback is provided during regression testing, as the scale of testing increases significantly due to frequent changes in the system under test and practice to continually execute large test suites [15]–[17]. In addition to prior research, the challenge of receiving faster feedback was highlighted during multiple workshops with software practitioners as part of our research. Developer productivity decreases when test suites grow and execution time becomes prohibitive due to slower feedback from testing cycles [15], [17], [18] and an overwhelming amount of test artifacts to analyze [19], [20]. For instance, developers at Google wait between 45 minutes to 9 hours to receive feedback from test failures [15]. Shahin et al. [21] concluded that CI practices begin to fail when developers are unable to obtain feedback from tests in a timely manner. Another study [22] reports that even when high-performance computers are used, Microsoft's regression tests for a single software product take several days. Similarly, for its 13K projects, Google's Test Automation Platform requires 150 million test runs [15]. Consequently, regression testing becomes a critical bottleneck in the pipeline increasing feedback cycles related to the quality of the developed product. Immediate feedback from testing cycles has been perceived to reinforced developers' sense of accomplishment and heightened their motivation [23].

Faster feedback from CI/CD testing cycles is significant, however the quality of the feedback provided and the developers' trust in it are just as significant [24]. The developer's productivity suffers when the test results, in addition to a longer execution time, are non-deterministic (i.e., cannot be trusted), known as test flakiness [25]–[28]. Developers submit code changes with the expectation that test feedback (i.e., test failures) will be associated with the code modifications. Unfortunately, rather than being the result of changes to the code, some test failures occur due to flaky tests. In the literature, the most common definition of a flaky test is: a test that exhibits both passing and failing outcomes when no changes are introduced into the code base [27]. Flaky tests are defined as *"tests whose outcome is not deterministic"*. Many large-scale software projects suffer from a high amount of test flakiness. For example, Google has reported that 80% of failing tests were due to flakes and only about 20% were actual regressions [29]. Another study confirmed 1 in 7 of the tests at Google sometimes fail due to test flakiness [30].

Overall, given the complexity of CI/CD, slower feedback to developers from testing cycles, combined with test flakiness (i.e., a lack of trust in test failures), can outweigh the perceived benefits of CI/CD, or at the very least be considered a significant waste during product development. In such situations, it is necessary to:

- Understanding/identifying the information needs of software practitioners in CI/CD.
- Adopt simple test optimization approaches that are realistic for use in contemporary CI/CD environments without adding too many technical requirements.
- Understanding/identifying perceived causes and automated root cause analysis of test flakiness, thereby providing developers with indications on how to resolve test flakiness, leading to an increase in developer trust in feedback.
- Understanding/identifying challenges in addressing information needs, providing faster feedback and trusting the feedback.

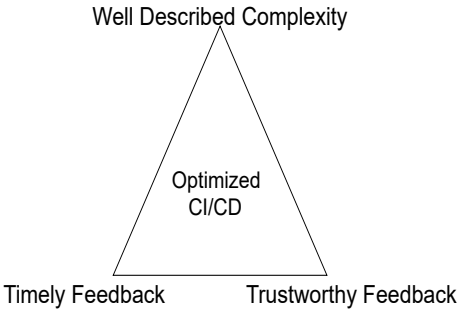


Figure 1.1: Factors to consider when achieving optimized CI/CD

1.1 Research Context

This work focuses on three factors to accomplish successful CI/CD optimization, as shown in Figure 1.1: *well-described complexity*, *timely feedback*, and *trustworthy feedback*. Given the project/industry circumstances (i.e., the nature of businesses), multiple interpretations of how these factors complement each other may exist. These three factors, however, were explored individually in separate investigations in this work.

All of the investigations in this work were carried out together with companies that produce embedded system software. All of the companies under investigation have achieved either a high level[1] of automation or a moderate level[2] of automation in CI/CD. To describe the *complexity* within CI/CD, we investigated different teams, including those in charge of source code, design, testing, and deployment. However, when it came to *feedback speed*, and *trust in feedback*, we interviewed teams that used automated regression testing due to the their willingness to cooperate and share their regression test suites. The regression test will be done by running all of the tests that were executed on the previous version on a new version and comparing the results. We used the definition of regression test, defined by IEEE: "A regression test will be performed for each new version of the system to detect unexpected impact resulting from program modifications" [31].

1.2 Research Questions & Dissertation Overview

As shown in Table 1.1, this work is organized around four main research questions (RQ1–RQ4), each of which has been further subdivided into multiple sub-research questions that have been answered in the papers. As illustrated in Figure 1.2, this work includes research outcomes as well as practical outcomes. Practical outcomes are in the form of open-source plugins or tools, whereas research outcomes are published in peer-reviewed venues. The following sections provide an overview of the research questions and studies that were conducted as part of this work.

Research Papers Addressing RQ1

The initial phase in this research endeavored to identify the information needs of software practitioners engaged in CI/CD and the mechanisms by which they address these needs.

[1]The CI pipeline support automation from commits to deployment with none or very little human intervention

[2]The CI pipeline support automation from commit to testing and human intervention is required for deployment

Table 1.1: Mapping of Research Goals/Questions to Published Papers

Reserach Questions/Research Goals	Papers
RQ1: What are practitioners' information needs in continuous integration and deployment and how do software professionals handle them?	
RQ1.1: What are practitioners' information needs in continuous integration and deployment?	Paper I
RQ1.2: To what extent are software tools utilized in industry to address the identified information needs?	Paper I
RQ1.3: Do practitioners assign priorities to identified information needs based on importance, frequency and effort?	Paper I
RQ2: To what extent, can different tools/techniques be utilized to achieve faster feedback and increase trust in the given feedback during regression testing in CI/CD?	
RQ2.1: How can DBT lever test feedback on CI pipelines?	Paper II
RQ2.2: What are the trade-offs in selecting and executing fewer test cases during software builds?	Paper II
RQ2.3: Is there a difference between diversity functions in selecting integration-level automated test cases?	Paper II
RQ2.4: How does diversity compare with other prioritisation approaches on CI feedback cycles concerning fault detection rates, coverage of features, and optimized execution time?	Paper III
RQ3: To what extent, can different tools/techniques be utilized to reduce test flakiness and increase trust in the given feedback during regression testing in CI/CD?	
RQ3.1: What factors do practitioners perceive as affecting the test flakiness?	Paper IV
RQ3.2: What are the root causes of test flakiness in closed source industry and how do professionals address test flakiness?	Paper IV
RQ3.3: Can perceived factors (i.e., test case size and simplicity) explain whether a test case is flaky or not?	Paper IV
RQ3.4: To what extent do different factors reveal test flakiness? Are there more effective combinations of factors, or are some factors better in isolation?	Paper V
RQ3.5: What type of information can MDFlaker reveal to developers to help understand the root causes of test flakiness?	Paper V
RQ3.6: What are the predictive accuracy of Naive Bayes, Support Vector Machine and Random Forest concerning flaky tests?	Paper VI
RQ3.7: To what extent the predicting power of machine learning classifiers vary when applied on software written in different programming language?	Paper VI
RQ4: What are the challenges/recommendations in adopting the tools/techniques mentioned in RQ2 and RQ3 and in answering information needs, identified in RQ1?	
RQ4.1: What challenges and effects impact adoption of DBT in CI pipelines?	Paper III
RQ4.2: What can we learn about the predictive power of test smells using machine learning classifiers mentioned in RQ 3.6?	Paper VI
RQ4.3: What challenges are faced by the practitioners that develop and maintain visualization tools for the software team?	Paper I
RQ4.4: What are the recommendations from practitioners that develop and maintain visualization tools for software teams concerning challenges, identified in RQ4.3?	Paper I

The RQ1 (i.e., Table 1.1) motivated the further investigations leading to Paper II-VI (see Figure 1.2). CI/CD includes a number of tasks (i.e., manual or automated) that must be performed on a daily basis by different stakeholders. Each day, these stakeholders are confronted with several questions relating to code, tests, builds, releases, and other aspects of product quality [11]–[14]. Software practitioners have different information needs depending on their particular roles and responsibilities [11]. Despite the fact that it is important to catalog and understand the questions and challenges faced by practitioners when attempting to answer those questions [13], little is known about (1) what information

needs practitioners have [11], [12] and (2) what information needs are unfulfilled in the field of software engineering [32]. These needs can help us better understand what tools, practices, and processes are important when we try to meet those needs [11], [12], [33]. We investigated the information needs associated with different stakeholders (i.e., RQ 1.1 as shown in Table 1.1). In paper I, we investigated to what extent software tools and techniques are utilized in industry to address the identified information needs (i.e., RQ 1.2). We documented the importance, frequency, required effort with respect to identified needs (i.e., RQ 1.3). In addition to identifying information needs, practitioners encounter a number of challenges while developing visualization tools (i.e., RQ 4.3). In this paper, we included recommendations from practitioners skilled in designing, maintaining, and offering visualization services to the software team (i.e., RQ 4.4).

Research Papers Addressing RQ2

During the investigation of Paper I, we realized that the demand for effective automated regression testing strategies grew in line with the increased use of iterative development processes and systematic reuse in software projects. Companies want to test their software frequently and efficiently while keeping a low cost [34]. Automated regression testing is, without a doubt, an expensive venture (i.e., 80% of testing cost is regression testing) [35]. Re-running these regression tests each time a change is made, no matter how small, will increase testing costs share the testing budget. It is worth noting here the challenges of energy consumption in terms of computer power and resources consumed during regression testing. For the companies under investigation, addressing these issues (i.e., feedback speed and testing costs) was of the utmost importance. As a result of our analysis in Paper I and the concerns expressed by software professionals during workshops about the lack of simple test optimization techniques in CI/CD, we decided to focus our research on test case optimization in automated regression testing. The term *simple* refers to affordable techniques that require little technical constraints and easily accessible data. Thus, the findings of RQ1 motivated the need for RQ2, as shown in Figure 1.2.

Numerous test optimization techniques have been proposed to minimize, select, or prioritize sets of test cases in order to gain faster feedback throughout testing cycles [36]. The industry, we collaborated with, brought in few constraints such as lack of sharing of software production code with outside researchers (due to business confidentiality) and limited information about internal working of system under test, limiting us to use black-box test optimization techniques. Aside from these constraints, the size of a test suite is increasing at a faster rate, indicating the need for test optimization techniques to receive faster feedback [17]. Recent results indicate that diversity-based testing (DBT) is a promising candidate for black-box test optimization in such situations [37] due to its failure detection and feature/requirement coverage. Diversity-based testing refers to a number of methodologies that use distance values to discover similar elements amongst test artifacts, such as specifications [20], [38], [39], test input or output [40], failure, and execution history [41], [42], among other things. We chose to focus on diversity-based testing (DBT) approaches since they are largely black-box with limited dependence on external artifacts (i.e., similar to our circumstances where we had lack of detailed information about system) and have demonstrated higher test efficacy when compared to other techniques. Particularly, diversity-based testing techniques outperform other black-box testing methods, such as combinatorial interaction testing and mutation-based testing [37].

By using DBT, we intended to reduce test cases, increase feature/requirements coverage and reveal faults during regression testing (i.e., RQ 2.1 in Table 1.1). Furthermore, we investigated what are the trade-offs in selecting fewer test cases during software builds (i.e., RQ 2.2)? This dissertation also documented if there is a difference between similarity functions in selecting integration-level automated test cases (i.e., RQ 2.3) as well as how

diversity compares with other prioritization approaches on CI feedback cycles concerning fault detection rates, coverage of features, and optimized execution time (i.e., RQ 2.4).

To address RQ2, two investigations (Paper II and III) were conducted. As indicated in Figure 1.2, this research effort resulted in two scientific outcomes (Papers II and III) and one practical outcome (the DBT tool)[3].

Research Papers Addressing RQ3

Developers submit code changes with the expectation that test failures will be associated with the code modifications. Unfortunately, rather than being the result of changes to the code, some test failures occur due to flaky tests. This concerns the software developers, in investigated companies (i.e., Paper I), when they talked about how much they have problem showing trust in the feedback from the testing cycles due to test flakiness.

We used a variety of approaches to address RQ3. For example, The first paper (i.e., Paper IV) was conducted to investigate practitioners' perceptions of test flakiness in a closed-source development environment (i.e., RQ 3.1). Perception is critical in the adoption of new techniques/processes by practitioners, as practitioners may choose local opinion over empirical data [43]. The scientific community has paid close attention to the necessity to understand practitioners' perceptions of software engineering. Researchers have investigated practitioners' perceptions of continuous integration [44], software design (e.g., code smells or exception handling) [45]–[48], software testing [49]–[51] and software quality [52]–[55]. Since flaky tests are a serious concern of software professionals, we decided to investigate practitioners' perceptions about what factors are perceived to affect test flakiness. Capturing these perceptions and comparing them to previous work will lead to a clearer understanding of test flakiness, which will prevent individuals from misunderstanding one other in the future. In addition to practitioners' perceptions, Paper IV investigated the root causes of test flakiness (i.e., RQ 3.2). The Paper IV reported whether or not the developers' perceptions match with what they have marked as flaky or not (i.e., RQ 3.3). The idea was to investigate what practitioners perceive and whether these perceptions are reflected in the test artifacts.

The most common method for fixing test flakiness is to re-run test cases. However, re-running test cases consumes resources (Google spends 2-16% of their testing budget on re-running tests [30]) and is still unreliable [56]. Furthermore, estimating the number of re-runs necessary to detect a difference between test results and the expected results is challenging [56]. Developers may be able to detect test flaws by rerunning the test numerous times and observing variations. However, practitioners' primary interest is to understand what causes test flakiness, requiring close attention to a frequently asked question by developers: *"why is this test case flaky?"*.

The goal is to develop novel tools and techniques to detect, predict, and automate root cause analysis of test flakiness, particularly to indicate the reasons for test cases being flaky. The factors identified in Paper IV, as well as some new factors, were investigated to see whether these factors played any role in exposing test flakiness (i.e., RQ 3.4). These factors were utilized to create an automated technique/tool (named as `MDFlaker`[4]) for detecting flaky tests as well as an automated root cause analysis of test flakiness (i.e., RQ 3.5). The goal for RQ 3.5 is to identify: what type of information `MDFlaker` can reveal to developers to help understand the root causes of test flakiness?. The `MDFlaker` consists of four factors: trace-back coverage, test smells, flaky frequency, and test case size. Each of the aforementioned factors provides significant information on its own, but when combined, they provide an

[3]https://gitlab.liu.se/azeah70/diversitybasedtesting
[4]https://gitlab.liu.se/azeah70/multifactorftdetector

aggregated score on test flakiness. We utilized these scores and the associated information to both prevent future test flakiness and to identify and remedy existing test flakiness. After acquiring information about the aforementioned factors in failed test cases, `MDFlaker` uses a machine-learning algorithm to classify whether failed test executions are flaky or not.

Another significant contribution (Paper VI) to the research of test flakiness is the prediction of test flakiness using machine learning (ML) approaches on test code. We used supervised ML classifiers to determine whether a test case is flaky or not based on the contents of a Python test case. We sought evidence that machine learning classifiers can predict flaky tests and that the results may be applied to test cases written in other languages. In addition, our unique contribution is to determine if test smells are accurate indicators of test flakiness. Through detailed investigation of false positives and false negatives, we were able to compile a list of test smells that are strong and weak predictors of test flakiness.

In summary, the RQ3 resulted in three research outputs (Papers IV, V, and VI) and one practical output (the MDFlaker tool), as illustrated in Figure 1.2.

Research Papers Addressing RQ4

Developing tools or techniques to address research problems comes with a multitude of challenges specific to such tools or techniques. Paper I, III and VI explore the factors that influence the adoption of tools and techniques proposed in RQ2 (i.e., DBT techniques) and RQ3 (i.e., machine learning classifiers for predicting test flakiness), as well as the challenges associated with addressing information needs via visualization tools in RQ1.

Number of studies [17], [57]–[59] highlight the effectiveness of test prioritization in continuous integration pipelines, but the literature is inadequate of insights into the challenges and guidelines associated with the adoption of diversity-based testing techniques in continuous integration pipelines (i.e., RQ 4.1). In fact, research findings are rarely discussed in terms of their practical impact [60], [61]. When it comes to regression testing techniques, different factors affect adoption, such as context suitability (e.g., system, domain, test process), the desired effects (e.g., improved test coverage or transparency in decisions) [62].

Similarly, in Paper VI, the limitations of using machine learning classifiers to predict test flakiness were discussed (i.e., RQ 4.2). Paper I looked into the various challenges that practitioners faced when designing visualization tools or selecting from a wide range of external (e.g., commercial or open-source) visualization tools (i.e., RQ 4.3). A good visualization tool improves decision making, ad hoc data analysis, user collaboration and communication, and return on investment [63]. Paper I listed recommendations from practitioners who are experts in developing, maintaining, and providing visualization services to the software team in Paper (i.e., RQ 4.4).

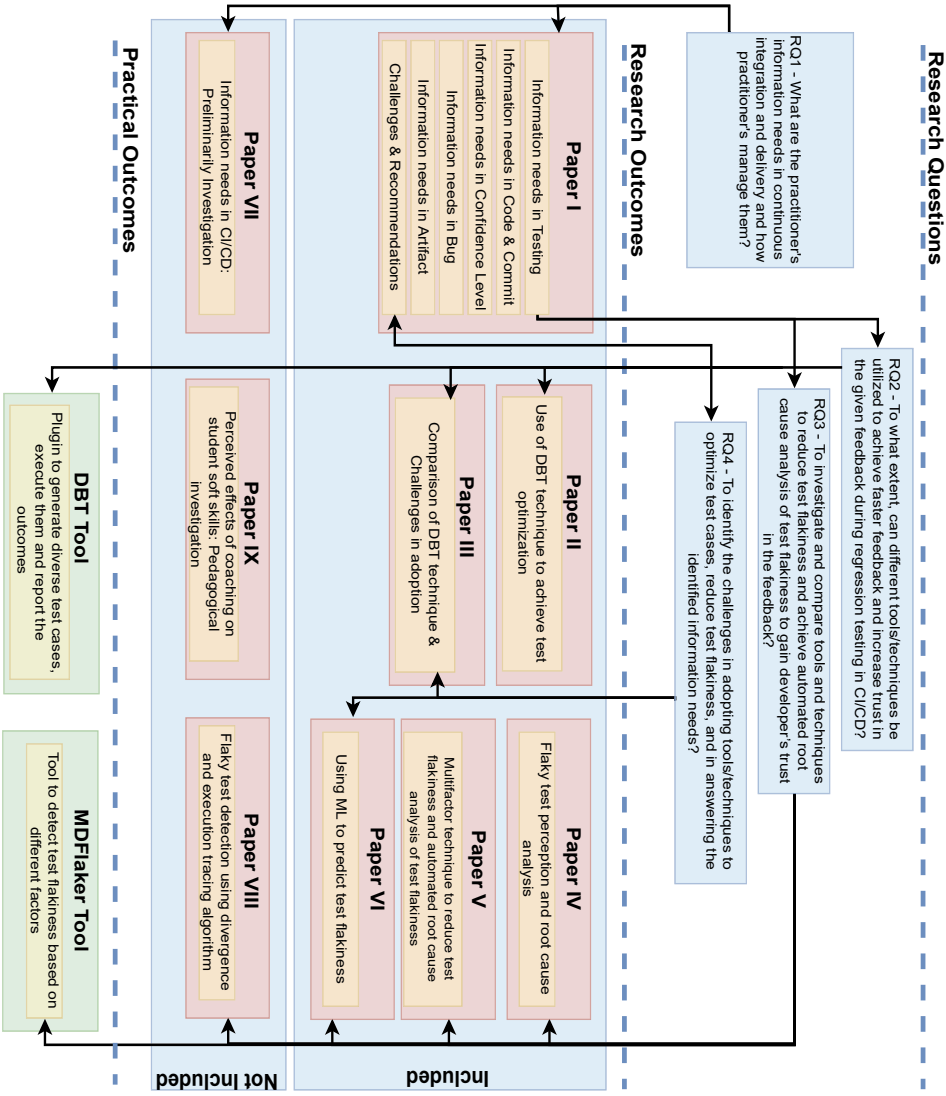


Figure 1.2: The links between research questions, research outcomes and practical outcomes

2 Personal Contributions

This dissertation contains articles that are the result of multiple authors' research and work. This section discusses how each article was influenced by the author's personal experiences.

2.1 Papers included in this work

Paper I - Data Visualization in Continuous Integration and Delivery: Information Needs, Challenges, and Recommendations

Publication details: Ahmad, A., Leifler, O., Sandahl, K.: Data visualisation in continuous integration and delivery: information needs, challenges, and recommendations. IET Soft. 16(3), 331– 349 (2022). https://doi.org/10.1049/sfw2.12030

The research design, data collection, determining analysis techniques, and result validations were all carried out by myself. Ola Leifler, Kristian Sandahl, and I worked together on the quantitative and qualitative data analysis. The resulting paper was written by myself and later reviewed by Ola Leifler and Kristian Sandahl.

Paper II - Improving Continuous Integration with Similarity-based Test Case Selection

Publication details: F. G. de Oliveira Neto, **A. Ahmad**, O. Leifler, K. Sandahl and E. Enoiu, "Improving Continuous Integration with Similarity-Based Test Case Selection", 2018 IEEE/ACM 13th International Workshop on Automation of Software Test (AST), Gothenburg, Sweden, 2018, pp. 39-45.

The research design, data collection, and data analysis for this article was a joint effort by myself and Francisco Gomes de Oliveira Neto. I implemented the programming scripts to parse the industry data with the guidance of Francisco Gomes to be used for analysis. Francisco Gomes wrote a paper. I, together with Kristian Sandahl, Eduard Enoiu and Ola Leifler, reviewed the paper.

Paper III - An Industrial Study on the Challenges and Effects of Diversity-based Testing in Continuous Integration

Publication details: A. Ahmad, F. G. de Oliveira Neto, E. Enoiu, K. Sandahl and O. Leifler, "An Industrial Study on the Challenges and Effects of Diversity-based Testing in Continuous Integration", submitted for publication

The research design, data collection, and data analysis for this article were a joint effort by all the authors. Francisco Gomes, Kristian Sandahl, and Eduard Enoiu collected the qualitative data, whereas I wrote the programming scripts to collect the quantitative data from test artifacts provided by the industry. I together with Eduard Enoiu transcribed the audio recording. The primary qualitative data analysis was conducted by myself and Eduard Enoiu. Later, Francisco Gomes, Ola Leifler, and Kristian Sandahl reviewed the analysis of qualitative data. I implemented the programming scripts to parse the industry data with the guidance of Francisco Gomes to be used for quantitative analysis. Francisco Gomes, Eduard Enoiu, and myself have written a paper together. Kristian Sandahl and Ola Leifler reviewed the paper.

Paper IV - Empirical Analysis of Practitioners' Perceptions of Test Flakiness Factors

Publication details: Ahmad, A, Leifler, O, Sandahl, K. Empirical analysis of practitioners' perceptions of test flakiness factors. Software Testing Verification Reliability. 2021; Vol. 31, no 8. https://doi.org/10.1002/stvr.1791

For this article, I was responsible for the idea design, data collection, and analysis of all of the information presented. I gathered all of the qualitative and quantitative information and carried out the entire analysis. I implemented the script to parse industry data in order to conduct quantitative analysis on it. Afterwards, Ola Leifler and Kristian Sandahl reviewed the findings of the qualitative and quantitative data analysis. I wrote a full-length paper. Kristian Sandahl and Ola Leifler were responsible for reviewing the paper.

Paper V - A Multi-factor Approach for Flaky Test Detection and Automated Root Cause Analysis

Publication details: A. Ahmad, F. G. de Oliveira Neto, Z. Shi, K. Sandahl and O. Leifler, "A Multi-factor Approach for Flaky Test Detection and Automated Root Cause Analysis," 28th Asia-Pacific Software Engineering Conference (APSEC), Taipei, Taiwan, 2021 pp. 338-348. doi: 10.1109/APSEC53868.2021.00041

For this article, I was responsible for the idea and design. The primary data collection and tool implementation was performed by the MS thesis student (Zhixiang Shi). The detailed analysis of all of the information presented were conducted by myself, MS thesis student and Francisco Gomes. Afterwards, Ola Leifler acted as mentor and Kristian Sandahl reviewed the findings of the qualitative and quantitative data analysis. I wrote a full-length paper. Kristian Sandahl and Ola Leifler were responsible for reviewing the paper.

Paper VI - An Evaluation of Machine Learning Methods for Predicting Flaky Tests

Publication details: A. Ahmad, O. Leifler and K. Sandahl, "An evaluation of machine learning methods for predicting flaky tests", in Proceedings of the 8th international workshop on quantitative approaches to software quality co-located with 27th asia-pacific software engineering conference (APSEC 2020), singapore (virtual), December 1, 2020, vol 2767, p. 37–46.

I was responsible for the idea design, data collection, and analysis of all of the information presented. I gathered all of the quantitative information and carried out the entire analysis. I implemented the script to parse open-source data in order to conduct quantitative analysis on it. Afterwards, Ola Leifler and Kristian Sandahl reviewed the findings of the data analysis. I wrote a full-length paper. Kristian Sandahl and Ola Leifler were responsible for reviewing the paper.

2.2 Papers not included in this work

Paper VII - Software professionals' information needs in continuous integration and delivery

Publication details: Azeem Ahmad, Ola Leifler, and Kristian Sandahl. Software professionals' information needs in continuous integration and delivery. In Proceedings of the 36th Annual ACM Symposium on Applied Computing (SAC '21). Association for Computing Machinery, New York, USA, 1513–1520. https://doi.org/10.1145/3412841.3442026

The research design, data collection, determining analysis techniques, and result validations were all carried out by myself. Ola Leifler, Kristian Sandahl, and I worked together on the quantitative and qualitative data analysis. The resulting paper was written by myself and later reviewed by Ola Leifler and Kristian Sandahl.

Paper VIII - Identifying Randomness related Flaky Tests through Divergence and Execution Tracing

Publication details: Azeem Ahmad, Ola Leifler, Erik Norrestam and Kristian Sandahl. 2022. Pending for presentation at 5th International Workshop on the Next Level of Test Automation

For this article, I was responsible for the idea and design. The primary data collection and tool implementation was performed by the MS thesis student (Erik Norrestam). The detailed analysis of all of the information presented were conducted by MS thesis in supervision of Ola Leifler and myself. The publication paper was produced, by me, by modifying the version of the MS thesis.

Paper IX - The Perceived Effects of Introducing Coaching on the Development of Student's Soft Skills Managing Software Quality

Publication details: A. Ahmad, K. Sandahl, en A. Barglund, "The Perceived Effects of Introducing Coaching on the Development of Student's Soft Skills Managing Software Quality", in Joint Proceedings of SEED 2021 & QuASoQ 2021 co-located with 28th Asia

Pacific Software Engineering Conference 2021, Taipei [Virtual], December 6, 2021, vol 3062, p. 22–29.

For this article, I was responsible for the idea. Kristian Sandahl, Aseel Barglund, and I worked together to create the design. All of the authors contributed to data collection and analysis. I wrote the resulting paper, which was later reviewed by Aseel Barglund and Kristian Sandahl.

2.3 Open-Source Tools

We developed two ready-to-use software tools for Paper II and Paper V for the companies. The first tool, named DBT plugin [1] implements the DBT technique on a Python test suite for both standalone applications and automated builds utilizing Jenkins, Travis CI, and similar tools. The tool is distributed under the *Apache 2.0 License*. The other tool, named MDFlaker [2] was provided under the *Apache 2.0* license and determines if test cases are flaky or not using four factors (as described in Paper V). These factors are used to train a KNN model, which is subsequently used to detect flaky tests.

[1] https://gitlab.liu.se/azeah70/diversitybasedtesting
[2] https://gitlab.liu.se/azeah70/multifactorftdetector

3 Method

This chapter provides an overview of the research process and methodology used in the papers included in this work.

3.1 Research Process and Methodology

For researchers, determining an appropriate research methodology is an important [64] and difficult [65] step in the research process. It is highly recommended to work with the research methodology systematically as suggested by Bailey and Handu [66]: *"a systematic process based on the scientific method that facilitates the identification of relationships and determination of differences in order to answer a question"*. People often link research studies in a way that leads to (1) figuring out what the problem is, (2) coming up with a solution, and (3) testing the solution. We used the same systematic approach, moving from problem identification to solution proposal and finally validation of the solution. We mapped the research studies to the systematic process as shown in Table 3.1.

In addition to the research process, choosing the right research design or method is just as important. Different purpose requires different research methods [67]. In accordance with Robson's classification [68], Runeson et al. [67] identified four different types of research purposes:

- *Exploratory* – determining what is happening, gaining new insights, and developing ideas and hypotheses for future research.
- *Descriptive* – portraying a situation or phenomenon.
- *Explanatory* – attempting to explain a situation or a problem, preferably but not always through the use of a causal relationship
- *Improving* – attempting to improve a particular aspect of the phenomenon being studied

We presented single- or multiple-case studies [69] in this dissertation, in which we investigated one or more phenomena in different cases. Because the objects of study are contemporary phenomena that are difficult to study in isolation, the case study methodology is well suited for many types of software engineering research [67]. Case studies allow for a more in-depth understanding of the phenomena under investigation as compared to controlled experiments [67]. Each case in a multiple case study should be selected carefully so that it either predicts a) similar results or b) contrasting results [69]. We picked (a) so that results from different cases would complement each other.

There are three different types of methods (i.e., quantitative, qualitative and mixed research) that can be used in the research [70]. Both qualitative and quantitative methods are used in this research in order to address our research questions. Combining qualitative and quantitative data tends to result in a more complete understanding of the study's phenomena [67]. Quantitative data consists of numbers and classes, whereas qualitative data consists of words, descriptions, pictures, diagrams, and so on [67]. Statistics are used to analyze quantitative data, whereas categorization and sorting are used to analyze qualitative data [67].

Problem Identification

All of the investigated studies (Papers I-VI) looked at different problems. As shown in Table 3.1, we used a variety of methods to investigate the problems, including a survey, site visits, workshops, interviews, and artifact evaluation. We explain each method below.

Survey: A survey is a type of empirical study in which data are gathered by asking people questions. It is used to obtain a numerical or quantitative description of a particular group of people [70]. We used a combination of closed-ended and open-ended questions to conduct the survey. We began the survey with closed-ended questions to familiarize participants with the issues. Closed-ended questions are a faster and less expensive method of surveying [70]. The survey included open-ended questions too which enabled practitioners to incorporate additional information and perspectives on a subject.

Site Visit: There is no comprehensive definition of a site visit available [71]. An evaluative site visit occurs when individuals with specific expertise and preparation travel to a site for a limited period of time to gather information about an evaluation object, either through personal experience or through the reported experiences of others, in order to prepare account of events addressing the site visit's purpose [71]. We visited a site to obtain such information during the work with Paper IV as shown in Table 3.1. However many workshops for data collection or validation, as explained in below section, were conducted at different sites.

Workshops: Workshops are a vital area of investigation in the fields of computer science [72]. We conducted workshops, in investigated studies, based on the principles proposed by Thoring et al. [72]. These principles are described below:

- *Focus Definitions* – develop a clear and concise research question and evaluation objectives. What is the workshop intended to accomplish? Is your primary objective to evaluate something or to create something?
- *Role Allocation* – assign roles to the various stakeholders or participants. Determine which roles should be informants for which types of data. Reduce the risk of researcher bias by separating mentoring from data collection
- *Triangulation* – combine various research techniques and compare data from multiple appropriate data sources.

Table 3.1: Papers and related research methods and focus

		Paper Focus													
		Problem Exploration					Proposing a Solution			Solution Validation					Type of Data
Papers	Research Methods	Survey	Site Visit	Workshops	Interview (S/U/SS)	Artifacts Evaluation	Interpretation	Tools	Techniques	Comparison With Literature/OSS OS	Comparison With Industry Tools	Interview (S/U/SS)	Workshop	Research Purpose	Quantitative Qualitative
Paper I	Multiple	-	-	-	S	✓	✓	-	-	-	-	-	✓	Exploratory	✓ ✓
Paper II	Multiple	-	-	-	S	✓	-	✓	✓	-	-	-	✓	Exploratory	✓ ✓
Paper III	Single	-	✓	-	U	✓	✓	✓	✓	-	✓	-	✓	Exploratory	✓ ✓
Paper IV	Multiple	✓	✓	✓	S	✓	✓	-	-	✓	-	-	✓	Improving	✓ ✓
Paper V	Single	-	-	-	-	✓	-	✓	✓	✓	-	-	✓	Improving	✓ -
Paper VI	Single	-	-	-	S	-	✓	-	-	✓	-	-	✓	Exploratory	✓ ✓

S: Supervised, *U:* Unsupervised, *SS:* Semi-supervised, *OS:* Open-source software

- *Transparency* – describe and publish your evaluation aims, methodology, selection criteria, participant data, workshop course, and workshop findings so that other researchers can replicate them.
- *Reflection* – make 3–5 key observations on the usefulness of your evaluation approach (not the outcomes). This will aid in the future improvement of the workshop assessment criteria and provide useful insights for other academics.

Artifacts Evaluation: In the software process, there are many things that make up artifacts [73]. Each one can be described by a measurement [73]. For example, a piece of software code could be evaluated by its size, functionality, complexity, modularity, and other factors [73]. As part of the data collection, we acquired a number of software artifacts. Several metrics (e.g., size, clarity, etc.) were utilized to evaluate the artifacts to be analyzed.

Proposing a Solution

All of the investigated Papers (Papers I-VI) proposed solutions in a different ways as shown in Table 3.1. According to Cambridge dictionary, the *interpretation* refers to *"an explanation or opinion of what something means"*. Papers I, III, IV and VI provided the solution in terms of interpretation by practitioners of the studied phenomenon. The *technique* describes a specific approach of doing something. The *tool* helps practitioner in applying the techniques.

Solution Validation

Validation was done in a variety of ways for each of the solutions presented in the research. For the purpose of providing objective evaluation, in two studies (e.g., Paper I and IV), the companies that participated in the validation studies were distinct from the companies who participated in the primary investigations. The usage of triangulation was employed in order to improve the precision of the validation. Triangulation involves approaching the studied object from many perspectives, resulting in a more comprehensive view [67]. There are four main types of triangulation that can be used, depending on the situation [74] as explained below:

- *Data (source) triangulation* – utilizing more than one data source or collecting the same data at various intervals
- *Observer triangulation* – using more than one observer in the study
- *Methodological triangulation* – combining various data collection approaches, such as qualitative and quantitative methods
- *Theory triangulation* – utilizing different theories or points of view

All of the above-mentioned triangulation techniques were applied in the investigations included in this dissertation. Data triangulation and methodological triangulation were employed in several studies such as a comparison with literature, comparison with open source software, comparison with industry tools, quantitative data, semi-structured interviews, and workshops,as shown in Table 3.1. Many more than one author served as an observer during data collection and validation, and theories from many perspectives (e.g., experiments conducted with open-source software or work of other researchers) were evaluated in different studies.

3.2 Research Methods Concerning Paper I

We conducted a multiple case study. Companies were labeled from A-F. We studied different business sectors to avoid biases. Both quantitative and qualitative data collection was performed during the work with Paper I as presented in Table 3.1. Data collection, analysis, and validation were carried out in two stages, as seen in Figure 7.1 of Paper I. During phase 1, we conducted individual interviews with 13 software practitioners from cases A-E to collect data. We recorded the conversations and took notes during the meetings with prior permission from the participants. We conducted 60 minutes long individual semi-structured interviews[1] to collect data from all participants.

We read all of the transcripts from the interviews and coded them according to *open coding* [75]. Each paragraph of the transcript of each case company was studied to determine what was said and each paragraph was labeled with one or more codes. After analysis, the initial codes were re-checked by other researchers (i.e., co-authors). Later, we conducted a physical workshop with 21 different participants from five industries to validate the results. For the data validation exercise, we selected different participants from the ones who had participated in the earlier data collection phase to avoid biases in the results.

The goal of phase 2 was to identify the challenges, therefore we conducted individual interviews with 5 software practitioners from cases A, B, and F. In addition, we requested participants to present the visualization tools that their teams have been using/developing. These participants were in charge of giving visualization support to developers and testers. To conduct, record, analyze, report, and validate the data/findings, a similar process (i.e., phase 1 above) was followed.

[1] https://tinyurl.com/y2uztk5w

3.3 Research Methods Concerning Paper II

We conducted a multiple case study to investigate the advantages of adopting diversity-based test case selection to utilize continuous integration by giving fast and meaningful feedback on integration test activities. This case study was carried out using information gathered from two companies. Since this was an exploratory case study as shown in Table 3.1, our goal was not to analyze the quality of their CI pipelines and activities; rather, we used our findings to explain each case and highlight specific aspects of their automated testing processes that may be improved. We visited organization to conduct semi-structured interviews with testers and learn about the connections between the CI pipeline and their testing efforts. Furthermore, practitioners exported test data from their CIs, such as test case specs, execution logs, and build information (time stamps, executed test cases, and so on), which we used for data collection.

3.4 Research Methods Concerning Paper III

We conducted a single case study in order to identify and analyze the challenges and consequences of implementing DBT in the workplace. In order to clarify communication between researchers and practitioners, we held a kick-off meeting (i.e., an unstructured interview) with practitioners at our industry partner, during which we elicited their context as part of the data collecting process to clarify communication. (Table 3.1).

In a nutshell, we evaluated regression testing within CI pipelines in a surveillance company in Sweden using data mining a test repository and conducting a focus group interview, among other methods. Our case study has been divided into two self-contained units of analysis, each of which will be used to address the research questions. Paper III is covered by the first unit of analysis (research A), which involves statistical analysis of DBT applied on archive data provided by the company. The other unit of analysis (research B) is based on the results of a focus group study conducted with practitioners. A large effort was dedicated for transcribing the audio recording as part of the data collection. Data collected from practitioners is grouped into topics using thematic analysis [76], which reveals limitations and possible benefits of incorporating DBT into their continuous improvement pipelines.

3.5 Research Methods Concerning Paper IV

We conducted a multiple case study, investigating 5 different cases. We collected data in all five cases through online and in-person workshops, site visits, and semi-structured interviews, as detailed in Table 3.1. Prior to the investigation, we used a Google form to determine case companies' interest in test flakiness. The survey is comprised of multiple choice questions and was developed following an analysis of prior literature on test flakiness.18 respondents from five different companies responded to the poll.We studied the findings of the online survey and developed open-ended questions for the online workshops. We encouraged discussion among workshop participants but did not expect them to reach an agreement.

Later, one of the authors made a visit to company A to interview two testers. The 180-minute interview was taped and transcribed onto an Excel file for analysis. The rationale for visiting the organization was to examine their testing procedure and documentation in person, as they claimed to have no flaky tests. The objective was to elicit tacit knowledge from the company's daily operations. During a visit, we received a complete test suite consisting of 1609 test cases from case A as well as 30 tests marked as flaky and 120 marked as non-flaky from case B. To investigate what practitioners perceive and whether these perceptions are reflected in the test artifacts, we ran an automated script on test artifacts.

We used open coding [75] to code the transcripts from the workshop and interviews. Each paragraph of the transcripts was labeled with one or more codes. Later, we analyzed all paragraphs from various companies to identify codes that were identical (axial codes). These codes were further classified according to the influencing factor and its effect, which indicated whether the factor was believed to increase or decrease test flakiness.

Following that, we held a physical workshop with ten participants from five different companies to validate the results. To prevent biases in the results, we chose individuals for the data validation exercise who were not involved in the earlier data collection phase. This workshop lasted 120 minutes and provided participants with a list of influencing factors along with their descriptions. The participants were asked to rank the elements' importance using the Likert scale (i.e., from 1 to 5, 'Strongly Agree' to 'Strongly Disagree'). Each factor was discussed with participants to ensure that they understood its significance. Along with assessing the importance of factors, participants were asked to rank their effect on test flakiness using a similar Likert scale.

3.6 Research Methods Concerning Paper V

We conduced a single case study using open-source data to assess which factors are more effective to predict flaky test executions. We selected three Github projects to run our case study based on different factors. We chose those three projects because they are reasonably large and belong to a variety of business domains. We selected a total of 212 versions that had at least one failing tests from the three projects, leading to a set of 2166 test failures. To compare our classification, we needed to know which test runs from the selected versions displayed flaky behavior. To do this, we checked out and executed each build 30 times in isolation (tests were ran in random orders in order to cover some of the flaky categories defined in literature). When a test's outcome changed during those 30 runs, we classified the test execution as flaky. These trials yielded failures from the test set, with 1372 flaky tests and 794 non-flaky tests.

We evaluated our approach and tool (MDFlaker) in a case study with open-source projects. Our goal was to assess which factors were more effective to identify flaky test executions, whether those factors help in identifying root causes for test flakiness and, lastly, how does MDFlaker compare with other existing tools that classify test flakiness

3.7 Research Methodology Concerning Paper VI

Using open-source data, we conducted a single case study. We created a script to extract the contents of twelve open-source projects' test cases. Following the extraction of the test case text, we investigated which of the test cases in our database had been mentioned as flaky by Lam et.al [77]. Following this mapping, we created a database that included the project name, test case name, test case content, and a label.

We used three machine learning classifiers named as Naive Bayes (NBC), Support Vector Machines (SVM), and Random Forest (RF). NBC is commonly used in classification and is well-known for producing outstanding results [78]. The appealing characteristic of SVM is that it eliminates the requirement for feature selections, making spam categorization simple and quick [79]. RF is an ensemble classification method (a technique that combines numerous base models to build an ideal predictive model) that can be used to solve challenges with data classification.

To evaluate the predictive accuracy of classifiers, accuracy as the only performance indices is not sufficient [80]. We used precision, recall, F1-score, ROC curve, false positives and false negatives [80].

4 Results Summary and Contributions

This chapter provides a summary of the findings from several investigations (i.e., Papers I-VI). We present answers to the research questions (i.e., RQ1-RQ4).

4.1 Answers to Research Question 1

This section summarizes the results of the RQ1. To facilitate reading, we have included the RQs below. Responses to RQs might serve as a starting point for organizations optimizing their CI/CD pipelines. The answers to RQ1 can assist organizations in identifying their specific CI/CD needs. After identifying needs, practitioners within the organization can prioritize the work in fulfilling them. The responses to RQ1 pertain to the "well described complexity" section of Figure 1.1.

RQ1: What are practitioners' information needs in continuous integration and deployment and how practitioners manages them?

RQ1.1: What are practitioners' information needs in continuous integration and deployment?

RQ1.2: To what extent are software tools utilized in industry to address the identified information needs?

RQ1.3: Do practitioners assign priorities to identified information needs based on importance, frequency and effort?

Information Needs - RQ 1.1

To answer RQ1, we identified a total of 27 information needs. The identified information needs have been classified as *testing*, *code & commit*, *confidence*, *bug*, and *artifacts*. *Testing*

Table 4.1: Importance, Frequency, Effort and Time With Respect To Information Needs

ID	Information Need	Importance	Frequency	Effort	Time
C1	How much confidence do we have in the release to deplo to the customers?	4,8	**4,9**	**5,0**	15-20
CC6	Is the given feature ready to release to customers?	4,5	**4,6**	**5,0**	15-20
B3	Is the bug fix ready to release to customers?	4,5	**4,6**	**5,0**	15-20
C2	How much confidence do we have in the test suite?	4,1	**4,9**	**5,0**	**>20**
C3	How much confidence do we have in stand-alone projects t be merged into the master branch/baseline?	4,1	**4,7**	**4,8**	**>20**
CC2	What is the status/health of new code changes?	4,1	**4,6**	**4,8**	15-20
CC4	Which change request does the specific commit implements	4,0	**4,7**	3,5	10-15
CC1	Does the final release to customers include my code?	4,0	3,7	2,5	5-10
T3	In which environment/machine do specific test cases fail?	3,8	**4,7**	**4,5**	>20
T7	Which test cases are flaky?	3,7	**4,7**	**5,0**	**>20**
CC5	Is the given feature implemented?	3,6	**4,6**	**4,5**	10-15
B1	Which bugs have been fixed in the specific release?	3,2	**4,3**	3,3	10-15
T5	What are the build/test results of my commits?	3,1	**4,7**	**5,0**	**>20**
CC3	Which requirement does the specific commit implements?	3,0	**4,7**	3,0	10-15
B4	Who broke the build?	3,0	3,5	**4,0**	**>20**
T6	What are the unstable areas of the code that require mor testing/attention?	2,9	**4,3**	**5,0**	**20**
CC8	How has my code affected non-functional properties of th product?	2,7	**4,6**	3,4	**>20**
A1	What tasks are pending in the pipeline for a long time?	2,6	2,7	**4,0**	15-20
B2	How many bugs are still open with specific release?	2,4	2,6	2,0	10-15
T1	Which test cases/suites have been run on which product?	2,2	**4,3**	**5,0**	15-20
T2	Which test cases/suites have been run on which branch?	2,1	**4,3**	**5,0**	15-20
T4	Which test suite' execution times have increased recently?	1,9	3,6	1,0	10-15
T8	What is a test execution history of a specific test case?	1,5	3,6	1,0	10-15
CC9	Which internal release notes have my comments/code?	1,5	1,9	**5,0**	**>20**
A2	When and why was this artifact created/modified?	1,3	1,7	3,5	15-20
A3	Who created this artifact?	1,3	1,7	3,0	15-20
CC7	How often does a specific employee deliver new code to th system?	1,0	1,1	**5,0**	**>20**

and *code & commit* each has eight associated needs as shown in Table 4.1. Three needs were identified in the *confidence* level category, four in the *bug* category, and five within *artifacts*. The details of the information needs are presented in Section 7.

How Practitioners Manages Information Needs - RQ 1.2

Table 4.2 shows how companies (A-E) deal with information needs, what tools they use to deal with them, and who is interested in them. *"In-house visualization"* refers to the visualization tools that have been built by the company's IT department to address the information needs. *"External visualization tool"* refers to situations wherein stakeholders can seek answers from the direct output of external tools or plugins. For example, answers for T_1 & T_2 can be directly sought through the default output from the Jenkin's test framework, or through the use of plugins with no manual intervention necessary. *"Manual inspection"* is a combination of internal/external tools and human intervention such as manually traversing the logs, sending emails to colleagues, or querying databases. We observed that only case A has in-house visualization tools that provide complete answers

to T_4, $CC_{1\text{-}4}$, $C_{1,3}$ and partial answers to T_3 and C_2 due to the fact that this company needs external tools to get a complete answer. On the other hand, ten information needs out of twenty-seven, such as $T_{5\text{-}6,8}$, $CC_{5\text{-}7,9}$, B_3, and $A_{2\text{-}3}$, can only be answered through manual inspections of output from tools. A general summary to increase readability about information needs, their stakeholders and how it is addressed is presented in Table 4.2.

Jenkins [81] is a widely adopted CI tool for addressing the identified information needs, followed by GitHub [1] and Jira [2]. Gerrit [3] was mentioned only once by case E to address CC_1. We asked participants about stakeholders that would be interested in knowing the answers to the identified information needs as represented in Table 4.2. We observed that none of the information needs belonged to just one stakeholder but several. Different stakeholders are connected to needs in different capacities. *"Compliance Authority"* is an external stakeholder that requires answers to certain questions before the product can be released to customers. This was only applicable to case E.

Information Need's Priorities Based on Importance, Frequency & Effort - RQ 1.3

Practitioners were asked to rank the information needs based on their importance, frequency of occurrence, and effort involved in addressing them prior to building or selecting an appropriate tool. Practitioners should prioritize information needs that are most critical and frequently requested in order to boost productivity and save time [11]. Eight (29%) information needs, as presented in Table 4.1, are marked as either "important" or "very important". Participants consider information needs such as $C_{1\text{-}3}$, $CC_{1,2,4,6}$, and B_3 significantly important. On the other hand, six (22%) information needs have been marked as either "good to know" or "of little importance". We did not see any correlation between the needs and classification. The needs cover different activities such as test execution time (T_4), test case life cycle (T_8), internal release notes (CC_9), artifact related question ($A_{2\text{-}3}$) and employee performance (CC_7).

When it comes to the frequency of answering the information needs, 17 needs (62%) were marked as either "frequently" or "very frequently" as represented by the bold text in Table 4.1. Three (11%) needs — which received a low ranking for importance — were marked as either "depends on context" or "rarely". All important needs (i.e., moderately to very important) were sought either "frequently" or "very frequently" in day-to-day to routines of the practitioners.

As far as effort to address information needs is concerned, 17 information needs (63%) were marked either as "difficult" or "very difficult". One can imagine the magnitude of difficulty practitioners faced due to the fact that all frequently sought needs were mentioned either as "difficult" or "very difficult". Only four (14%) needs were marked either "easy" or "very easy".

As seen in the Table 4.1, we also asked how much time practitioners spent looking for information. It took more than 10 minutes to answer 24 (87%) information needs, whereas just 3 needs were answered in under 10 minutes. All of the "frequently" labeled needs took more than 10 minutes to address.

[1] https://github.com
[2] https://www.atlassian.com/software/jira
[3] https://www.gerritcodereview.com/

Table 4.2: Strategies, Tools and Stakeholders With Respect To Information Needs

Label ID		Current Handling Approach			Tools/ Strategies Used					Teams				
		In-house Tools	External Tools	Manual Inspection of Tools' output	Jenkins	GitHub	Gerrit	Jira	Manual (e.g., send email, call, etc.)	Development	Testing	Project Management	Release Team	Compliance Authority
T1	Which test cases/suites have been run on which product?		ABCD		ABCD					X	X	X	X	X
T2	Which test cases/suites have been run on which branch?		ABCD		ABCD					X	X	X	X	X
T3	In which environment/machine do specific test cases fail?	A		ABDE	BDE					X	X	X		X
T4	Which test suites' execution times have increased recently?	A								X	X	X		
T5	What are the build/test results of my commits?			ABDE	ABDE				ABDE	X				
T6	What are the unstable areas of the code that require more testing/attention?			BCE					BCE	X	X	X		
T7	Which of the test cases are flaky?		ABCDE	ABCDE	ABCDE				ABCDE	X	X	X		
T8	What is a test execution history of a specific test case?			DE	DE	DE			DE	X	X	X		
CC1	Does the final release to customers include my code?		A E	BDC		E			BDC	X	X	X		
CC2	What is the status/health of new code changes?		A BCE	BCE	BE				C	X	X	X		
CC3	Which requirement does the specific commit implements?		A BDE		BDE			BDE		X	X	X		
CC4	Which change request does the specific commit implements?		A BDE		BDE			BDE		X	X	X		
CC5	Is the given new feature implemented?			ABDE		AB			ABDE		X			
CC6	Is the given feature ready to release to customers?			ABCDE	AB				ABCDE	X		X		
CC7	How often does a specific employee deliver new code to the system?			AD					AD			X		
CC8	How has my code affected non-functional properties of the product?		B	BD					BD			X		
CC9	Which internal release notes have my comments/code?			AD					AD	X				
C1	How much confidence do we have in the release to deploy to the customers?		A	BDE	B	BDE			BDE	X	X	X	X	X
C2	How much confidence do we have in the test suite?		A A	BDE	ABDE				BDE	X	X	X	X	X
C3	How much confidence do we have in stand-alone projects to be merged into the master branch/baseline?	A		BDE		BDE			BDE	X	X	X		
B1	Which bugs have been fixed in the specific release?	A		ABDE			A		BDE			X	X	
B2	How many bugs are still open with specific release?		ABCDE	ABCDE				ABCDE	ABCDE		X	X	X	X
B3	Is the bug fix ready to release to customers?			ABCDE	ABC			ABC	ABCDE	X		X	X	
B4	Who broke the build?		BDE	BDE	BDE	BDE			ABCDE	X	X	X		
A1	What tasks are pending in the pipeline for a long time?		AB	CDE	AB	CDE			AB	X	X	X		
A2	When and why was this artifact created/modified?			CDE		CDE			CDE	X	X	X		
A3	Who created this artifact?			CDE		CDE			CDE	X	X	X		

4.2 Answers to Research Question 2

This section contains a summary of the RQ2 results. The RQs are displayed below to improve reading. These RQs were covered in Papers II and III. This RQ is about improving feedback speed during automated regression testing. The responses to RQ2 are for the "Timely Feedback" part of Figure 1.1.

RQ2: To what extent, can different tools/techniques be utilized to achieve faster feedback and increase trust in the given feedback during regression testing in CI/CD?

RQ2.1: How can DBT lever test feedback on CI pipelines?

RQ2.2: Is there a difference between diversity functions in selecting integration-level automated test cases?

RQ2.3: How does diversity compare with other prioritization approaches on CI feedback cycles concerning fault detection rates, coverage of features, and optimized execution time?

DBT to Lever Test Feedback on CI/CD Pipelines & Differences in Diversity Functions - RQ 2.1 - 2.2

In this case study we consider three distinct attributes in a test case: *test requirements*, *test dependencies*, *test steps*. As a functional assessment of the system under test, a test requirement serves as an approval criterion. Test requirements include things like validating a system need (e.g., a hardware element should respond properly to network commands). Dependencies on functionality for testing are also known as test dependencies, and they are not included in the approval criteria. Test dependencies refers to test requirements that must be present to test the specific test requirements. For example, to test the test requirement `A`, the SUT must contain test requirement `X`, thus making `X` as a test dependency. Test requirements and dependencies are the necessary components for a test to be carried out successfully. Test steps are the plain language description of user activities and expected outcomes that a tester creates. Because these tests are at the integration level, the natural language description is eventually converted into test code that can be executed.

We evaluated diversity-based selection techniques in terms of the percentage of number of requirements covered by the test cases, dependencies, and steps covered as we reduced the number of test cases in steps of 5% of the total number of test cases. This enables us to observe how coverage changes when we run fewer test cases. As a control group, we utilized random selection (RDM), whereas the remaining three levels represent various similarity functions used in published studies [39], [40], [42], [82]: Normalised Levenshtein (NL), Jaccard Index (JI) and Normalised Compression Distance (NCD).

When it comes to test requirement and dependency coverage, both the JI and NL techniques outperform RDM and NCD by offering higher and more sustainable coverage when the number of test cases is reduced. It is worth noting that both techniques can eliminate up to 85% (931 /1096) of test cases while still covering all distinct test criteria (Figure 4.1). In other words, even if a developer is working locally on a single test requirement, she can submit her changes to the CI server and receive feedback from all test requirements while only running 15% of the test suite. NL and JI outperform other selection techniques when test dependencies are considered (Figure 4.2). It is important to note that while each test covers a single requirement, each test case may have multiple dependents. Surprisingly, NCD has performed worse than RDM. When employed on small strings, compression can

impair NCD, according to Feldt [40]. Our dataset's test requirements are short strings, which may favor NL because the edit distance provides a fine-grained similarity value.

Since the time falls linearly as we remove test cases (see Figure 4.3), the first conclusion to be drawn is that we cannot differentiate the strategies when time reduction is concerned. On the other side, we can see the benefits of DBT in CI pipelines by integrating findings from time (Figure 4.3) and coverage (Figure 4.1-4.2). Note that the full test suite runs overnight since developers cannot wait an average of 225 minutes for testing results. However, as shown in Figure 4.3 shows that only 15% of the test cases can be successfully executed in 17 minutes by a developer.

Figure 4.4 illustrates our findings for test steps in automotive company. Regrettably, the case company could only share information on test steps at this time. Nonetheless, the results are informative, especially when compared between the two units of analysis. NL beats the existing similarity functions by offering high and durable test step coverage, similar to the results from Surveillance Company. Notably, test cases for Automotive company are also integration-level test cases written in natural language. Figure 4.4 demonstrates that even after removing 75% (1479 /1972) of the most identical test cases, NL can consistently cover 99% (1082 /1093 steps) of distinct test steps. When compared to random selection, more than half (58 percent = 634 steps) of unique test steps are discarded. Test steps, on the other hand, are short strings that further impair NCD's performance.

Test selection strategies in general are susceptible to the risk of discarding test cases, because ideally, all test cases should be executed [36]. Our approach is based on the assumption that the whole test suite is run nightly. As a result, failures should be reported the following day so that developers can debug and, if required, revert their changes.

Results from both of our case studies show that similarity functions behave differently depending on the type of property being tested in the test case. As a result, NCD and NL both did worse than JI and NL because the data from all three properties (steps, requirements, and dependencies) were all small strings of text. If we use a longer string (like a concatenation of all three properties), we think NCD will work better.

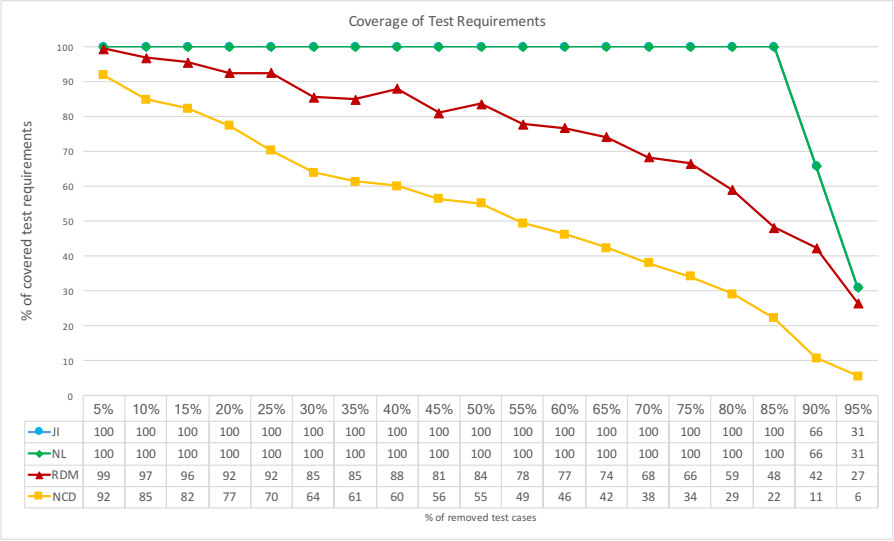
DBT Comparison with an Industry Tool - RQ 2.3

We looked at three different ways of prioritizing (diversity, failure history, and time) to see how they affected coverage, failure detection rates, and the time it took to run tests. Failure history and time prioritizes were used by a Swedish company in their day-to-day activities.

All of the techniques, except for the time-based prioritization (i.e., Figure 4.5), work in a linear way when it comes to time. In other words, there are no specific advantages to using one method over the other because most tests take the same amount of time to run. In terms of average percentage of fault detected (APFD[4], putting failure rate first is the best way to get the best results. All failures are quickly found by first running the test cases that, based on the history, fail more often. For example, 90% of the faults are found after only 5% of the tests have been run.

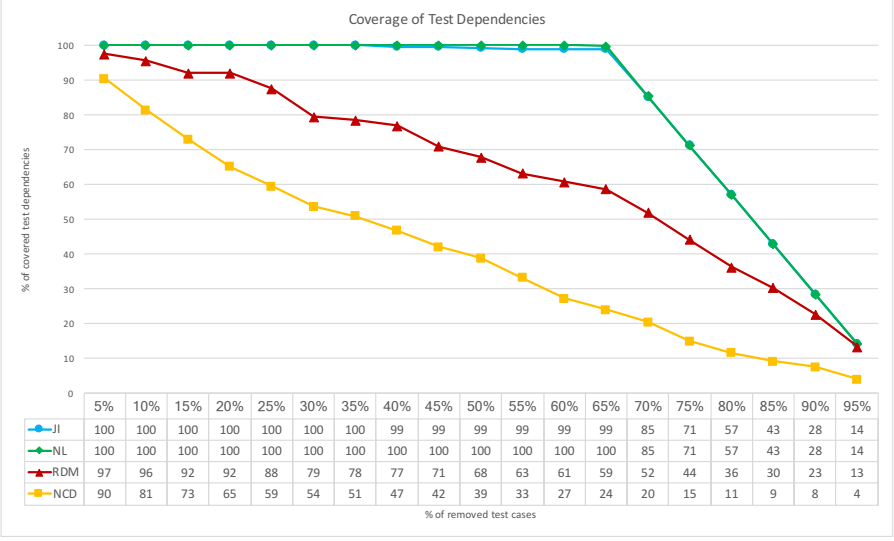
Diversity has the best results when it comes to mean coverage. With only 15% of the tests covering all of the features. This isn't the case with APFD. Failure Rate is very similar to other techniques. There is a bigger difference in coverage between the techniques if you have a smaller budget. It is better to use Random prioritization than both Failure Rate and Time, if you have a budget that allows 30% or 40% of test cases to be executed. Random

[4]APFD can provide faster feedback on the SUT, and help software programmers begin locating and resolving defects earlier than might otherwise be possible



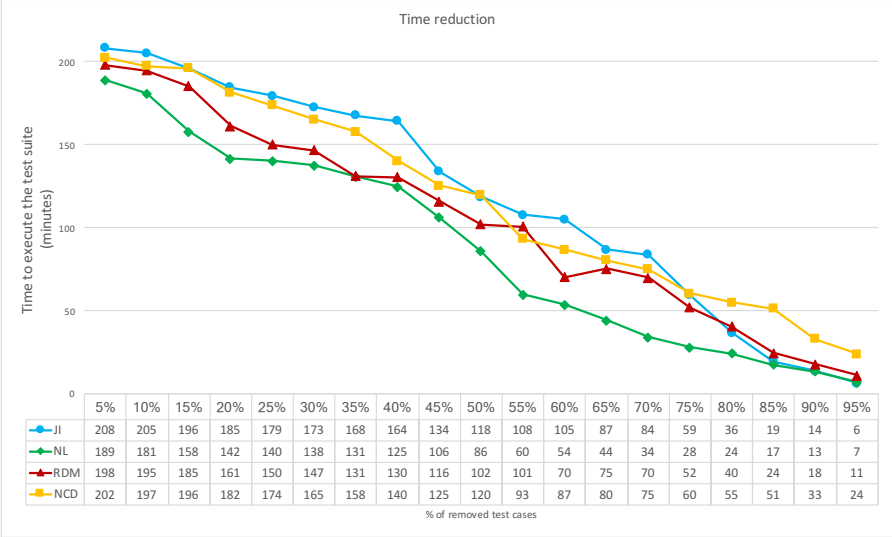
	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%
JI	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	66	31
NL	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	66	31
RDM	99	97	96	92	92	85	85	88	81	84	78	77	74	68	66	59	48	42	27
NCD	92	85	82	77	70	64	61	60	56	55	49	46	42	38	34	29	22	11	6

Figure 4.1: Results from test selection on test requirements using data from the surveillance company



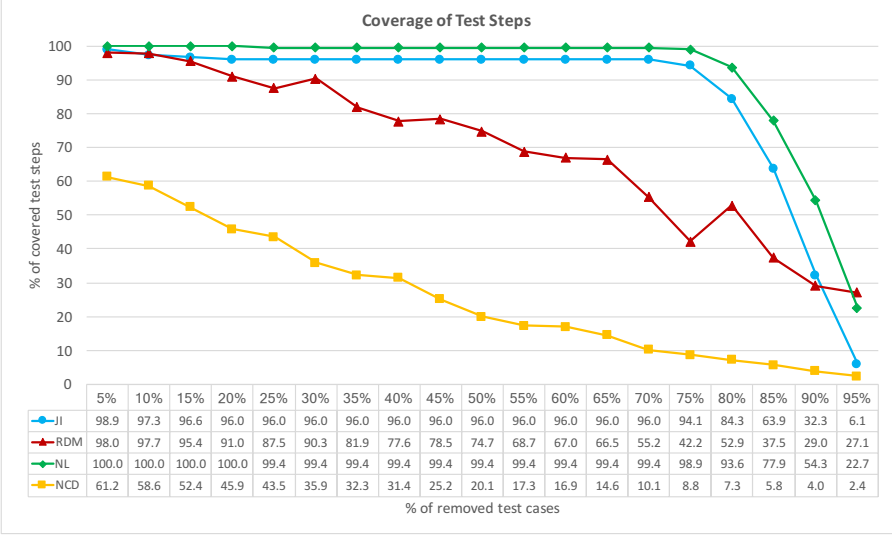
	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%
JI	100	100	100	100	100	100	100	99	99	99	99	99	99	85	71	57	43	28	14
NL	100	100	100	100	100	100	100	100	100	100	100	100	100	85	71	57	43	28	14
RDM	97	96	92	92	88	79	78	77	71	68	63	61	59	52	44	36	30	23	13
NCD	90	81	73	65	59	54	51	47	42	39	33	27	24	20	15	11	9	8	4

Figure 4.2: Results from test selection on dependence coverage using data from the surveillance company



	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%
JI	208	205	196	185	179	173	168	164	134	118	108	105	87	84	59	36	19	14	6
NL	189	181	158	142	140	138	131	125	106	86	60	54	44	34	28	24	17	13	7
RDM	198	195	185	161	150	147	131	130	116	102	101	70	75	70	52	40	24	18	11
NCD	202	197	196	182	174	165	158	140	125	120	93	87	80	75	60	55	51	33	24

Figure 4.3: Results from test selection on time reduction using data from the surveillance company



	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%
JI	98.9	97.3	96.6	96.0	96.0	96.0	96.0	96.0	96.0	96.0	96.0	96.0	96.0	96.0	94.1	84.3	63.9	32.3	6.1
RDM	98.0	97.7	95.4	91.0	87.5	90.3	81.9	77.6	78.5	74.7	68.7	67.0	66.5	55.2	42.2	52.9	37.5	29.0	27.1
NL	100.0	100.0	100.0	100.0	99.4	99.4	99.4	99.4	99.4	99.4	99.4	99.4	99.4	99.4	98.9	93.6	77.9	54.3	22.7
NCD	61.2	58.6	52.4	45.9	43.5	35.9	32.3	31.4	25.2	20.1	17.3	16.9	14.6	10.1	8.8	7.3	5.8	4.0	2.4

Figure 4.4: Results from Automotive company regarding coverage of test steps as we select fewer test cases

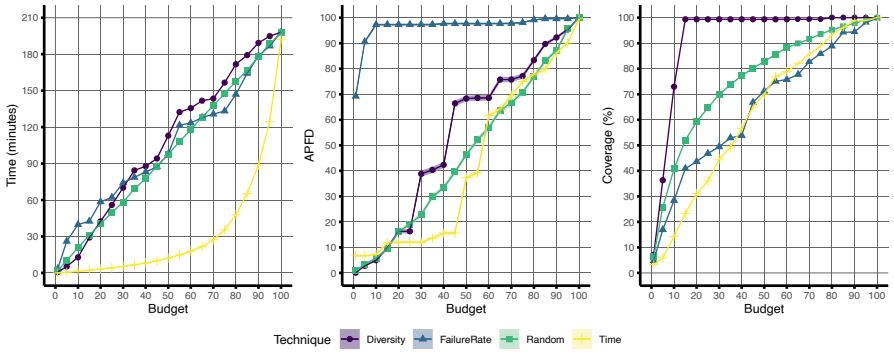


Figure 4.5: Results for mean time, APFD and coverage of different prioritisation techniques considering an increasing test execution budget.

can cover 20% more features than both techniques. Test suites that were prioritized by time did the worst in both measures, so we don't think it should be used in CI cycles.

4.3 Answers to Research Question 3

This section contains a summary of the RQ3 results. The RQs are displayed below to improve reading. These RQs were covered in studies IV, V, and VI. These studies are conducted in response to one of the identified information needs labeled T_7 in Table 4.1. We investigated the issue of test flakiness using various techniques and methods. The responses to RQ3 concern the "Trustworthy Feedback" part of Figure 1.1.

RQ3: To what extent, can different tools/techniques be utilized to reduce test flakiness and increase trust in the given feedback during regression testing in CI/CD?

RQ3.1: What factors do practitioners perceive as affecting the test flakiness?

RQ3.2: What are the root causes of test flakiness in closed source industry and how do professionals address test flakiness?

RQ3.3: Can perceived factors (i.e., test case size and simplicity) explain whether a test case is flaky or not?

RQ3.4: To what extent do the chosen factors reveal test flakiness? Are there more effective combinations of factors, or are some factors better in isolation?

RQ3.5: What type of information can MDFlaker reveal to developers to help understand the root causes of test flakiness?

RQ3.6: What are the predictive accuracy of Naive Bayes, Support Vector Machine and Random Forest concerning flaky test detection and prediction?

RQ3.7: To what extent the predicting power of machine learning classifiers vary when applied on software written in other programming language?

Practitioners' Perceptions about Test Flakiness- RQ 3.1

Paper IV found 19 factors that professionals perceive affect test flakiness. These perceived factors are divided into four categories: test code, system under test, CI/test infrastructure, and organization-related. We concluded that some of the perceived causes in test flakiness in closed source development are directly related to non-determinism, whilst others involve

different aspects, such as a lack of suitable test case properties, deviations from established processes, and ad hoc decisions. Six perceived factors have been documented in the literature as characteristics of effective test cases, three perceived factors are related to a system under test, eight perceived factors concern CI/Test infrastructure, and two perceived factors are organizational-related.

Practitioners shared whether the perceived effect of the perceived factor can either increase or decrease test flakiness. We grouped these perceived factors based on their effect as represented by Figure 4.6. Fifteen perceived factors out of nineteen are claimed to decrease test flakiness whereas four factors to increase test flakiness.

Root Causes of Test Flakiness in the Investigated Companies - RQ 3.2

We explored the core causes of test flakiness in terms of the test smells encountered by professionals, in addition to capturing the various aspects of managing/identifying test flakiness. Many studies [83]–[86] have investigated the relationship between test smells and test flakiness. We provided the list of test smells in the survey [5] that is known to relate to test flakiness. 'Asynchronous wait' and 'configuration & dependency issues' have been mentioned as a major root cause of test flakiness by all participants, as represented by Figure 4.7 (h). Other major concerns expressed by almost half of the participants included 'GUI', 'I/O operations', 'Randomness', 'Test order dependency', and 'Time'. Four of the test smells, as shown in Figure 4.7, were not encountered by any of the participants: 'Concurrency,' 'Resource leak,' 'Floating-point operations,' and 'Unordered collections.'

Practitioner's Perceptions vs Test Artifacts - RQ 3.3

To see if the developers' expectations were in line with what they had labelled as flaky or not, we examined the test artifacts that were readily available. To get a statistical assessment of the size and simplicity of the test cases, automated scripts were used. We counted the number of assertions in the test case to represent the term simplicity, where a low number of assertions signals a simple test case. The size of the test case represents the number of lines of code without counting comments. Figure 4.8 presents the box-plot for test case size and simplicity within two companies with respect to flaky or non-flaky tests. A correlation between test flakiness and test case size was found in company B, according to our findings. Figure 4.8 shows that the non-flaky tests in company B have a lower number of lines than the flaky tests. The median for non-flaky tests in cases A and B is 24 and 35 respectively. The median in flaky tests in case B is 95 which is far more higher than from non-flaky tests. For company A, we did not have flaky tests. However, the lines of codes in non-flaky tests in company A are on similar level to the lines of codes in non-flaky tests in company B, as shown in Figure 4.8. We suspect that this is one of the reasons why company A has not suffered flakiness.

Case B revealed a similar distinction between test flakiness and test case simplicity. As illustrated in Figure 4.8, non-flaky tests in case B have fewer assertions than flaky tests. The median for non-flaky tests in cases A and B was 2, while the median for flaky tests in case B was 5. Case A has one to two assertions in each non-flaky test case, with the exception of a few circumstances when the number of assertions surpasses fourteen (i.e., outliers in Figure 4.8). Again, we believe this is the explanation for the lack of flakiness in case A. The authors concluded, based on a data set of studied instances, that two of the perceived factors (i.e., test case size and test case simplicity) have an effect on test flakiness.

[5] `https://tinyurl.com/yxpw3vgu`

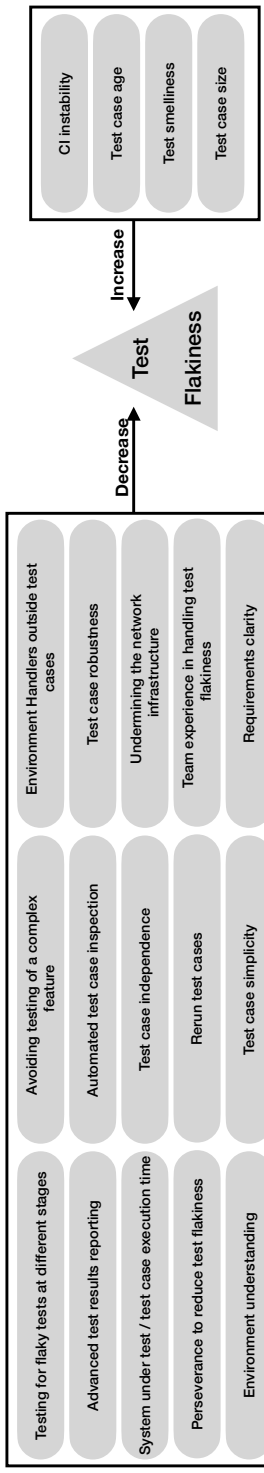


Figure 4.6: Tentative Mapping of Relationships Based on Perceptions.

Figure 4.7: Survey Results Presented with Questions in Each Facet

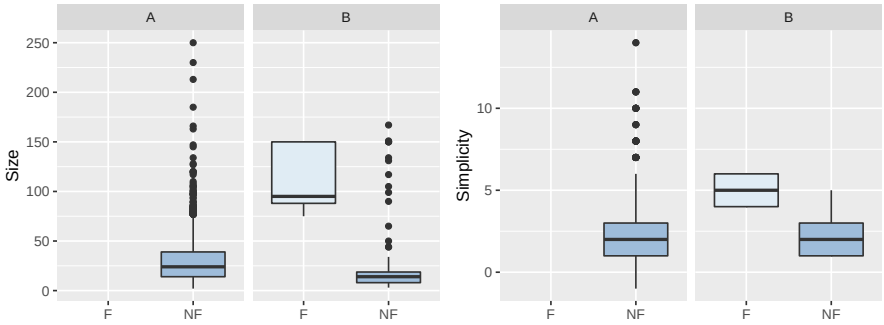


Figure 4.8: Size (i.e., lines of non-comment codes) and simplicity (i.e., number of assertions) in the test artifacts within companies A and B with respect to flaky and non-flaky tests. Note: case A did not provide any marked flaky tests and claimed that all tests are non-flaky

Multi-Factor Approach for Flaky Test Detection and Automated Root Cause Analysis - RQ 3.4 - 3.5

Paper V presented and tested MDFlaker: a multi-factor technique for flaky test identification on open-source data. This technique does not only detect flaky test cases, but also explains why they are flaky. MDFlaker is composed of the following factors: trace-back coverage, test smell, flaky frequency, and test case size. MDFlaker employs a machine-learning method (k-Nearest Neighbour or KNN) to classify whether failed test executions are flaky or not after acquiring information about the aforementioned factors in failed test instances. Each factor employed by MDFlaker is summarized here.

Trace-back coverage: This factor is a modified technique based on Bell's differential coverage [87]. Trace-back is a simple technique that uses information from test execution logs to flag a failed test case as flaky if it did not execute on a code change.

Flaky Frequency: MDFlaker keeps a history for each test case where the outcome changes without any changes to the codebase. The tests that switch their results without any changes to the codebase on multiple occasions in the past have a greater flaky frequency and are more likely to be flaky [88], [89].

Test Smells: Many studies empirically show the relationship between test smells and test flakiness [83], [84], [86]. In order to leverage from such relationships, `MDFlaker` detects and maintains a database of test cases and the number of test smells present in the test cases.

Test Case Size: MDFlaker maintains a database of each test case's lines of code. There is evidence that the size of the test case adds to test flakiness [89].

Each of the aforementioned factors provides significant information on its own, but when combined, they provide an aggregated score on test flakiness. We utilize these scores and the associated information to both prevent future test flakiness and to identify and remedy existing test flakiness.

MDFlaker's architecture is depicted in Figure 4.9. Test size and smell statistics are extracted from GitHub's source/test code. The trace-back coverage is estimated using the Travis build history and GitHub modifications. Finally, the flaky frequency is computed using the build

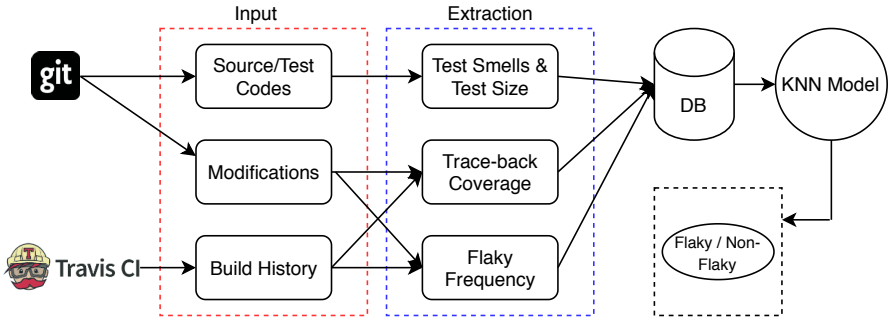


Figure 4.9: A high level architecture of *MDFlaker*.

history and changes on GitHub. Every piece of data is kept in a database. We used the KNN model to predict if failed test executions are flaky or not.

According to the findings of Paper V and as we can see in Table 4.3, combining all factors often results in high precision, accuracy, and recall. By comparing the results, in Table 4.3, we see that the different factors are more effective in identifying flaky tests than a random approach, despite the high proportion of flaky tests in our evaluation subset (circa 55%). However, because different combinations produce different results, we propose that the availability of artifacts should be the primary motivator for selecting the combination of factors (e.g., practitioners should avoid using frequency if history information is not available or scarce). In detecting true positives and true negatives, trace-back and frequency complement one other. Because test smells produce more false positives but less false negatives than other factors, they can be coupled with history-based factor due to their strong recall value. Size is not a good predictor since it produces many false positives and false negatives.

While rerunning test cases can show which test cases are flaky, it provides insufficient or no evidence concerning the root causes of test flakiness. MDFlaker gives various forms of information (e.g., CSV files or graphs) regarding either individual test cases or a whole test suite as part of the reporting flaky tests. This information can help developers in resolving test flakiness or at the very least provide an idea of where to begin looking. The tables, created by MDFlaker, contain detailed information about the test executions classified as flaky, such as test case name, the number of test smells, types of test smells, line number, and file name as presented in Figure 4.10 and 4.11. By visualizing additional information connected to failures, practitioners are more equipped to debug [17].

Flaky Test Detection & Prediction: Predictive Accuracy of Machine Learning Classifiers RQ 3.6

As a preventative measure, Paper VI examined the viability of applying machine learning (ML) classifiers for flaky test prediction in a `Python` project. The purpose of this study is to compare the predicted accuracy of three machine learning classifiers: Naive Bayes (NBC), Support Vector Machines (SVM), and Random Forests (RF). We compared our findings to an earlier investigation [90] of similar machine learning classifiers for `Java` based projects. We examined whether test smells are effective indicators of test flakiness. As developers must have confidence in machine learning classifier predictions, they want to know which types of input data or test smells result in the most false negatives and positives.

The receiver operating characteristic (ROC) curve [91] for NBC with Laplace smoothing, labeled as NBL with varied thresholds, is depicted in Figure 4.12 (A) (i.e., from 0.0 to 1.0).

Table 4.3: Overview of accuracy, precision and recall for each combination of factor and a random classification as a comparison baseline. As a reference, we also included the number of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN) based on the `MDFlaker`'s classification.

Row	Factor	FN	FP	TN	TP	Accu.	Prec.	Rec.
1	frequency	81	75	160	247	0.72	0.76	0.75
2	size	38	163	72	290	0.64	0.64	0.88
3	smells	29	222	13	299	0.55	0.57	0.91
4	trace-back	19	73	162	309	0.83	0.80	0.94
5	size, frequency	77	70	165	251	0.73	0.78	0.76
6	smells, size	69	121	114	259	0.66	0.68	0.79
7	smells, frequency	77	72	163	251	0.73	0.77	0.76
8	trace-back, size	27	76	159	301	0.81	0.79	0.91
9	trace-back, smells	18	76	159	310	0.83	0.80	0.94
10	trace-back, frequency	36	60	175	292	0.82	0.83	0.89
11	size, smells, frequency	77	63	172	251	0.75	0.79	0.76
12	trace-back, size, frequency	31	56	179	297	0.84	0.84	0.90
13	trace-back, size, smells	31	64	171	297	0.83	0.82	0.90
14	trace-back, smells, frequency	58	43	192	270	0.82	0.86	0.82
15	trace-back, smells, frequency, size	33	48	187	295	0.85	0.86	0.89
16	random	166	124	111	162	0.48	0.56	0.49

We conducted various experiments using various training and testing data sets, including 50/50, 60/40, 70/30, 80/20, and 90/10. The ROC curve compares the sensitivity and specificity of classifiers, which aids in organizing and visualizing their performance [91]. Sensitivity, also called true positive rate, refers to the advantage of correctly predicting flaky tests, whereas specificity, sometimes called false positive rate, refers to the expense of accurately classifying non-flaky tests as flaky tests. When a false positive occurs, developers must expend effort and time determining that the error is due to a classifier error and not a flaky test case. The best aim for the ROC curve is to advance vertically from the origin to the top left corner (higher true positive rate) as quickly as possible, as this allows the classifier to reach all true positives at the expense of committing a few false positives. In Figure 4.12 (A), the diagonal line shows the method of randomly predicting the outcome. Any classifier that appears in the lower right triangle performs worse than a random guessing and we can see that NBL lies in the upper left triangle. As illustrated in Figure 4.12 (A), NBL ceased producing positive classifications (i.e., flaky test predictions) around the 0.76 - 0.87 threshold. After 0.87, it commits a higher rate of false positives. Prior to doing further experiments, we tuned several parameters in NBL, SVM, and RF. We do not aim to include the results of all tests, as they were conducted solely to determine the optimal parameters. The rest (i.e., simple NB, SVM with radial and sigmiod kernels) were not included in further experiments and discarded. Figure 4.12 (A-E) provides comparisons of

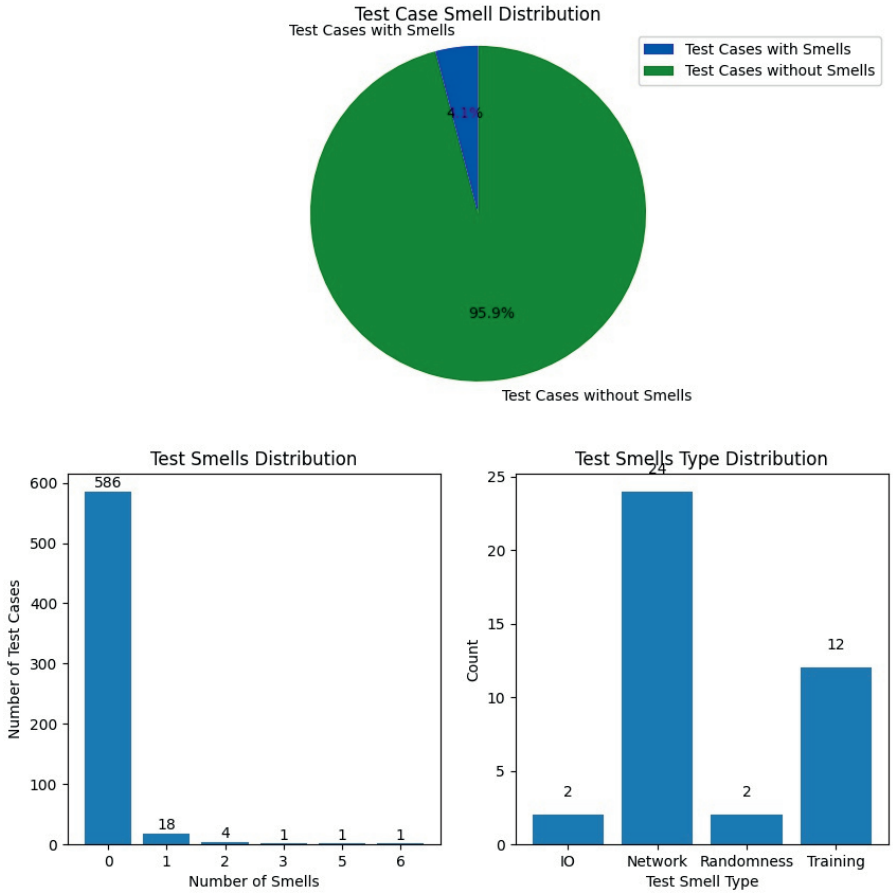


Figure 4.10: Images generated by *MDFlaker* to present different types of information about test smells in test cases.

NBL, SVM-Linear and SVM-Poly (i.e., different kernels) for accuracy, precision, recall and F1-score. All classifiers had high accuracies of between 93 and 96 percent. Although the difference is not significant, NBL outperformed SVM. It can be noticed that NBL precision is increasing (Figure 4.12 (C)) with the gradual decrease in recall (Figure 4.12 (D). With a precision of 65 percent, the NBL indicates that 35% of what was labeled as flaky was not. NBL also has a lower recall than SVM-Linear. SVM-Poly performs poorly in terms of precision and recall, as expected given the non-polynomial nature of the input data set, which is ideally suited for image processing, but the linear kernel performs better for text classification. The F1-score (Figure 4.12 (E)) is the harmonic mean of precision and recall. Due to the widespread problem of class imbalance in text categorization, the F1-score is beneficial and instructive [92]. Although NBL has a lower F1-score than SVM-Linear, it is a suitable alternative since it performs better with short documents, as in our case, the training test case consists of six to fifteen lines of code [93]. In comparison to SVM-linear, NBL has a greater precision but a poorer recall.

Test Case	Number of Smells	Smell type	Tip	Location	Path
test_lex_attrs_l	6	Network	URL	58	D:\spacy\tests\lang\test_attrs.py
test_lex_attrs_l	6	Network	URL	58	D:\spacy\tests\lang\test_attrs.py
…….	…..	…..	……	…..	……
test_issue1506	5	Training	For in range	24	D:spacy\tests\regression\test_issue1501-2000.py
test_issue1506	5	Training	For in range	26	D:\spacy\tests\regression\test_issue1501-2000.py

Figure 4.11: A Snapshot from CSV File: Detailed Description of Test Smells, their Type, Test Case Name, Line Number and File Name from the Project Under Investigation

In comparison to decision trees, NBL, and SVM, RF produces lower classification errors and higher F1-scores. Precision, which is what we are most concerned with, is typically higher than that of SVM and NBL. Additionally, the authors [80] concluded that RF outperforms NBL and SVM. Figure 4.13 presents the performance of RF with respect to selected metrics. *mtry* denotes the number of variables randomly selected as candidates for each split, whereas *ntree* denotes the number of trees to grow. Because there is no method to determine the ideal *mtry* and *ntree* values, we tested with various choices, as illustrated in Figure 4.13. The *mtry* has a direct effect on precision and recall. With an increase in *mtry*, precision decreases and recall increases; this is an undesirable scenario. The ideal value of *mtry* is 5, which results in increased precision and decreased recall regardless of the number of trees. The modification in *mtry* had no effect on accuracy, but as previously said, we are interested in precision as well as accuracy.

As can be shown, RF with *mtry = 5* and *ntree=250* outperforms all other classifiers in terms of precision only. RF has a precision of greater than 90% and a recall of less than 10%. We did not attain a high degree of precision (i.e., greater than 90%) in all classifiers. NBL provides unexpected results although it holds a good reputation in terms of detecting spam emails [94]. In comparison to NBL and SVM, RF has several distinguishing characteristics, including: 1) it can work with thousands of different input features without requiring feature deletion; 2) it approximates critical classification characteristics; and 3) it is extremely resilient to noise and outliers [95].

Predicting Power of ML Classifiers with Respect to Other Languages - RQ 3.7

Due to the fact that we used machine learning classifiers on test case code, paper VI specified specific features of test cases that contributed to the test cases being flaky or not flaky. Table 4.4 displays the 20 features with the biggest information gain, as well as their frequency in flaky and non-flaky tests. We classified the features according to the categories proposed by Luo et al. [83]. The top feature "conn" was present in 1361 flaky tests and in 15 non-flaky tests. This feature is connected with external connection to input/output devices and falls under the "IO" category. The second most common feature is "double," which appeared in 1190 flaky tests and 12 non-flaky tests classified as "IO," followed by "floating point operations." The fourth most common feature, "tabl," was related to table formation during runtime for database queries, and it appeared 1150 times in flaky tests and 52 times in non-flaky tests.

When we compared our findings to those of Pinto et. al [90], we noticed two differences. First, the top 20 features in both research projects are substantially different. Only two features, "tabl" and "throw," highlighted with a star (*) in Table 4.4, were found to be

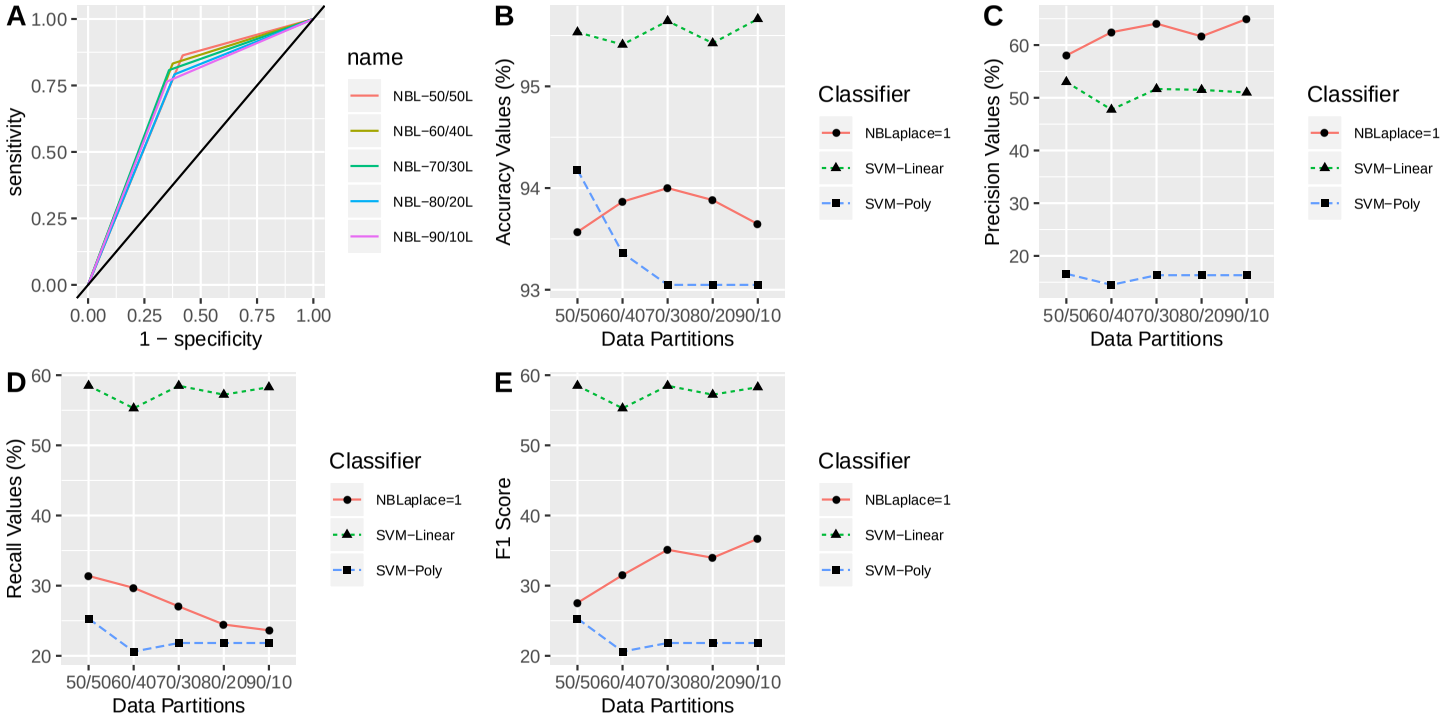


Figure 4.12: Performance comparison among classifiers. (A) represents the ROC curve of NBL classifier with different data partition and probability score. (B-E) represents the accuracy, precision, recall and F1-score of different classifiers with a different data partition, respectively.

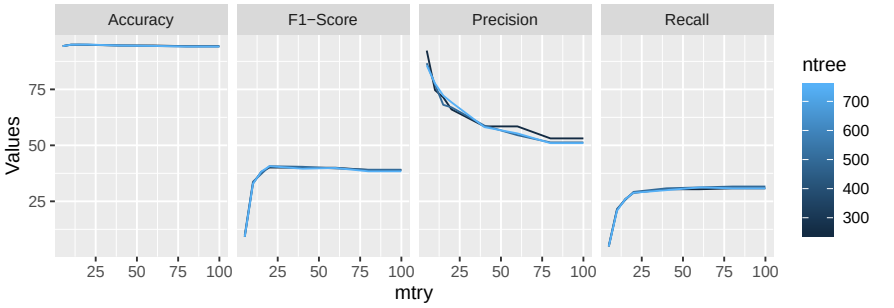


Figure 4.13: Performance of RF with different parameters (i.e., number of trees and mtry).

Table 4.4: Top 20 features and assigned category

Features	#FT	#Non-FT	Assigned category from Luo et. al [83]
conn	1361	15	IO
double	1190	12	floating point
tabl*	1150	54	-
rsnext	500	22	Unordered collections
for	241	15	-
jdbcassertfullresultsetr	900	0	IO
messag	1101	87	concurrency
null	334	88	-
sclose	360	0	IO
select	1080	15	IO
sgettransactioncommit	700	15	IO
expr	162	2	-
tcommit	134	5	-
true	700	11	-
epsilon	383	0	floating point
fail	269	13	-
jdbcassertcolumnnames	366	0	IO
throw*	592	49	-
rsclose	300	0	IO
row	161	3	-

similar in both studies. However, we discovered that the majority of our features were connected to the "IO" output category, as shown in Table 4.4, which supported the findings of Pinto et al., who stated that "all projects showing flakiness are IO-intensive." Second, we have lower precision, recall, and f1-score than Pinto et al., except in one case where random forest provided 0.92 precision, thus observing different results contrary to what Pinto et al. claimed: "Although the analyzed projects are largely written in Java, we do not expect major variations in the results if another object-oriented programming language is chosen instead, because some terms may be shared among them [90]."

4.4 Answers to Research Question 4

This section contains a summary of the RQ4 results. The RQs are displayed below to improve reading. These RQs were covered in studies I, III, and VI. The responses to RQ4 concern the "Well Described Complexity" section of Figure 1.1.

Challenges and Effects in the Adoption of DBT - RQ 4.1

Paper III listed three distinct themes as part of identifying challenges in adopting DBT and perceived effects of DBT. These themes are (1) testing activities, (2) organizational factors, and (3) test infrastructure. Various sets of challenges and effects were categorized under these themes to give insights such as more challenging areas or aspects in which DBT can be more effective. Figure 4.14 summarizes all themes with corresponding challenges and perceived effects of DBT. "Testing theme" refers to the challenges and effects of regression test activities at the case company, such as how to handle different levels of testing or tacit knowledge from experienced practitioners. The theme "organizational factors" focuses on the main challenges and effects of DBT in how engineers collaborate as a team, such as how they maintain test artifacts (test case templates, test execution logs, or design guidelines). These artifacts demonstrate how DBT can assist practitioners in producing better tests or improving test repository upkeep. The theme ‘testing infrastructure‘ discusses some of the challenges which are connected to the instrumentation behind the automated testing tool chain used in the company.

RQ4: What are the challenges/recommendations in adopting the tools/techniques mentioned in RQ2 and RQ3 and in answering information needs, identified in RQ1?

RQ4.1: What challenges and effects impact adoption of DBT in CI pipelines?

RQ4.2: What can we learn about the predictive power of test smells using machine learning classifiers mentioned in RQ1?

RQ4.3: What challenges are faced by the practitioners that develop and maintain visualization tools for the software team?

RQ4.4: What are the recommendations from practitioners that develop and maintain visualization tools for software teams concerning challenges, identified in RQ4.3?

Participants, in Paper III, noted that, in terms of technical challenges, the test repository itself should be automated to eliminate the burden associated with artifact inconsistencies (C3,E2). Shorter cycles with selected testing should include a form of confidence score in association with their CI process to highlight the risk associated with not running all tests at a single build (C6, E6). In terms of adoption, the main technical documented challenges are related to the compatibility of an extendable automated test instrumentation and the test prioritizing technique (C7,E7). The majority of the adoption challenges are associated with test processes and artifacts, such as assisting practitioners in decision making (C1,C2,E1), understanding how the technique is employed in their process (C4,E4), or giving compliance checks and transparency to team members (C5,E5).

Challenges Concerning Predictive Power of Machine Learning Algorithm in Determining Test Flakiness - RQ 4.2

During the work with Paper V, we discovered that it is not just the frequency of test smell that causes a flaky test case, but also its coexistence with the test class code (i.e., tear up and

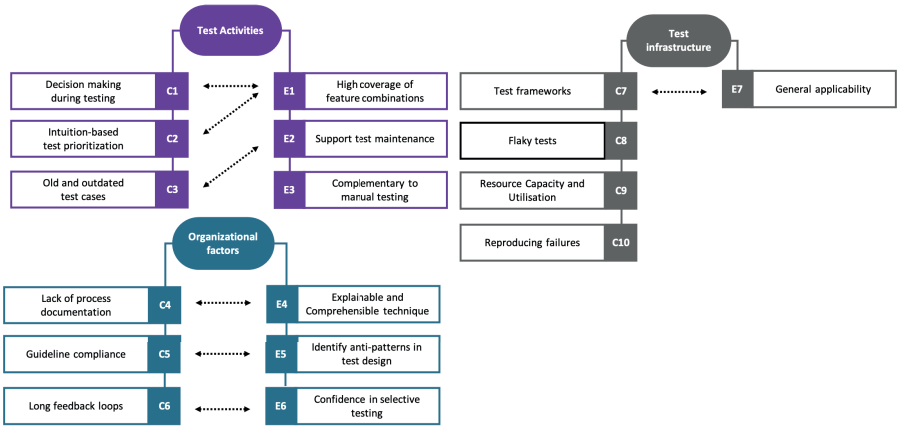


Figure 4.14: Overview of themes identified during our thematic analysis. Each theme includes a set of challenges and perceived effects related to the adoption of DBT.

Table 4.5: Test Smells as Strong and Weak Predictors Together with Source of their Existence

Test Smell Category	Prediction Category	Test Case	Test Class	Operating System	External Libraries	Hardware
Async wait	Strong	[✓]	-	-	-	-
Precision (float operations)	Strong	[✓]	-	-	-	-
Randomness	Strong	[✓]	-	-	-	-
IO	Strong	[✓]	-	-	-	-
Unordered Collection	Weak	[✓]	[✓]	[✓]	-	-
Time	Weak	[✓]	[✓]	[✓]	-	[✓]
Platform	Weak	[✓]	[✓]	[✓]	[✓]	-
Concurrency	Weak	[✓]	[✓]	[✓]	-	-
Test order dependency	Weak	[✓]	[✓]	[✓]	[✓]	[✓]
Resource Leak	Weak	[✓]	[✓]	[✓]	-	-

down code etc.,) or external circumstances such as operating systems or certain products. For example, The test smell *'Conditional Test Logic'* as mentioned in [96] refers to nested and complex *'if-else'* structure in the test case. Depending on which branch of *'if-else'* is executed, the system under test may require specific environment settings. Failing to set the environment, during different executions, will flip the test case outcome, thus making it flaky. We come up with a list of test smells that are strong or weak predictors of test flakiness, as shown in Table 4.5. Strong predictors are those test smells that were present in both true positive and negative cases, whereas weak predictors were present exclusively in false negative and positive instances. Although test smells classified as weak predictors in this study are still useful for identifying test flakiness, they are ineffective with machine learning classifiers because they require additional information such as the operating system on which they are running and whether or not specific configurations should be deployed. Test smells categorized as strong predictors are extremely valuable for machine learning

classifiers since they exist independently of the test case function and do not require further information.

Challenges & Recommendations in Addressing Information Needs through Visualization Tools - RQ 4.3 - RQ 4.4

Paper I looked into the challenges practitioners experienced while creating visualization tools or selecting from a vast range of external (commercial or open-source) visualization tools. A good visualization tool increases decision making, ad hoc data analysis, user collaboration and communication, and return on investment [63]. The investigation's goal is to identify practitioners' challenges when building visualization tools. We mapped challenges to identified information needs. The paper also compiled a list of practitioner recommendations for designing and maintaining visualization software. We discovered eight challenges and ten recommendations across all investigated organizations.

Challenges:

- **Visualization Tools Development & Maintenance:** Developing and maintaining visualization tools as business products presents a significant amount of difficulty. Unfortunately, due to the difficulty of deciding whether to develop an in-house solution or to investigate online resources (i.e., open-source, commercial off-the-shelf, etc.), visualization tools receive insufficient attention and resources.
- **Information Quality & Trust in Tooling:** It's difficult to earn the trust of developers and testers in visualization tools developed by other teams while also protecting the integrity of the data being shown.
- **Tool Creators vs Tool Users:** It is critical to establish a connection between tool users and tool creators, which is a significant problem. The absence of communication between two distinct user categories has an effect on the feedback loop.
- **Effort vs Worth:** Practitioners face challenges in persuading decision-makers at all levels to invest in the development of diverse visualization tools for different teams and to make it worthwhile.
- **Confidentiality vs Transparency:** One of the difficulties is striking a balance between confidentiality and transparency. When answering some queries, it is essential to verify if users are authorized to receive such information.
- **Different CI Work Flows:** Within large organizations, various teams work on a variety of different projects. Each team is free to choose their own continuous integration pipeline from a number of activities. Configuring visualization software for a variety of continuous integration workflows is challenging.
- **Result Interpretations:** When another team constructed the tool, the primary challenge for engineers is interpreting the results displayed on the screen. If the source of a result cannot be traced, it might be interpreted in a variety of ways.
- **User Interface Complexities:** During the release process, project managers want to know information about the product/software, such as how many test cases failed, what the release's confidence level is, and so on. Selecting which data should appear on the screen and how much flexibility we should have in tracing its origins is a difficult task.

Recommendations

R1 Developers will be more likely to trust the tool if it provides information about (1) the source of the information, (2) the age of the information, and other information quality attributes.

R2 To avoid becoming a legacy system, the visualization tool should incorporate specifics such as requirements specifications, design documents, and so on, in the same way that traditional product development activities do.

R3 It is recommended to create a basic architecture upon which other visualization tools can be constructed. Teams made use of a variety of open-source applications (e.g., GitHub, Gerrit, Jenkins, etc.), thus developing a base protocol similar to Eiffel [10] is encouraged.

R4 The credibility of a tool grows over time.They [developers] will trust it if they use it regularly.

R5 To simplify the work required to configure visualization systems, it is preferable for software teams to build identical and systematic CI pipelines (e.g., following the company's rules).

R6 Maintain a repository of software visualizations to reduce redundancy and increase re-usability.

R7 Bespoke development of visualization tools should be discouraged to reduce the risk of superfluous tools

R8 Any suggestion for a change in visualization requirements should be evaluated in real time by the appropriate change management teams.

R9 Depending on the needs and structure of the company, resources such as CI architects or maintainers should be increased. To support a quality product, there should be a balance of workload between people who develop business products and those who supply support services.

R10 Users of visualization tools should provide regular input to visualization tool developers in order for future decisions to be successful and efficient. Furthermore, these regular exchanges would limit the possibility of results being misinterpreted.

4.5 Key Contributions

The following are the key contributions in the form of research and practical contributions. Practitioners who are struggling to address the complexity of the CI/CD should begin by identifying the information needs. Taking it a step further, practitioners can use the methodologies and tools given in this dissertation, such as DBT and MDFlaker, to optimize CI/CD in terms of improving testing feedback time and confidence. These contributions are useful for both researchers and practitioners since they provide a systematic way to making adjustments rather than making changes randomly and hoping for the best. We list the key contributions below:

- **Information Needs:** The initial contribution to research is a list of information needs in CI/CD. This list contains 27 frequently asked questions on continuous integration and continuous delivery by software practitioners.
- **Perceived Factors that Affect Test Flakiness:** We identified 19 factors that professionals perceive affect test flakiness. These perceived factors are divided into four categories: test code, system under test, CI/test infrastructure, and organization-related.

- **Evaluation of DBT Technique/Tool:** Another contribution is to conduct a detailed examination of DBT techniques using industry artifacts and tools. The techniques are developed and made available as an open-source tool for further research and application.
- ***MDFlaker* Technique/Tool:** Additionally, we developed a novel technique and tool for reducing test flakiness, as well as an automated root cause analysis for test flakiness. The tool is open-source.

5 Discussion, Implications & Future Work

Papers I through IX contain extensive discussions of the investigations. The focus of this chapter, on the other hand, is on the synthesis of our papers combined together with its practical and research application.

5.1 Information Needs & Data Visualization - RQ 1 and RQ 4

The Impact of Complex CI/CD pipelines on Information Needs

Developing software products with CI/CD necessitates not only building, testing, and packaging software for installation, but also doing so on a regular basis. These frequent activities (building, deploying etc.) raise new information needs that require investigation. The prior work, in the field of information needs identification, was limited to single stakeholders (i.e., developers, testers, or project managers) and a single activity (i.e., development, software evolution, or software configurations). Few studies have been reported that identify the information needs as well as how much effort is required to address those information needs in the settings of CI/CD. Somewhere along the way, in addition to identifying information needs, it is equally important to investigate *if the company would be able to answer the very same questions with less effort without CI/CD?* Or *are these qualitatively new questions that can only be asked once CI/CD practices are in place?*

Identifying information needs is a first step. The other difficult task is to achieve common understanding of the specific terms used in describing software artifacts [97] particularity software features [98] and confidence in software artifacts [97]. Similarly, the top 3 needs in Table 4.1 might require the common understanding of the terms *confidence*, *realease-*

readiness or *feature*. Software professionals should make an effort to comprehend the concepts by explicitly asking questions such as: *What constitutes as acceptable confidence for a software release?* or *What information is required to achieve acceptable confidence in a software release and from what sources?*. Similar questions can be raised to understand *What does the feature mean?* or *Is secure communication or standard compliance a feature?*. These types of fundamental details will refine each information need further, thus creating a similar level of understanding among team members. The difficulty increased when practitioners tried to automate the process of defining/achieving `confidence` or `realease-readiness` by asking a question: *Can confidence or release decision be achieved through automation without human intervention?*. It is equally important to investigate the tust in artifacts as we do among team members.

We discovered that the term confidence in SUT is typically restricted to the number of test cases passed during a given time period. The greater the number of passing test cases, the greater the confidence in the SUT and the test suite. Concurrently, we have seen that practitioners struggle not to view this factor as a single indicator of SUT confidence. We also learned that the phrase "release-readiness" is assessed at impromptu meetings. A small group of practitioners from several teams (such as the product owner, test leader, and release manager) meet every two weeks or once a month to decide if the latest version is ready for release or not. To clarify these terms for our future work, we plan to do further research with the investigated companies. Furthermore, `trust` [1] is defined as "confident reliance on the character, skill, strength, or truth of someone or something." As previously stated, the quantity of test cases determines practitioners' confidence in the quality of SUT. Practitioners tend to lose this confidence if the outputs of those test cases cannot be trusted. Practitioners are unsure if trust can be quantified or should be relied solely on intuition. Clarifying phrases like confidence and trust will also help to eliminate the blame-game mentality in organizations.

The Challenges of Data Visualization in CI/CD

Along with cataloging information needs in CI/CD, it is important to identify the means through which those requirements can be addressed. Table 4.2 provides different means (i.e., in-house visualization tools, external visualization tools, manual inspection etc.,) used by practitioners for addressing information needs. Visualization tools were actively sought out by practitioners, who viewed them as potentially useful (i.e., see Table 4.2). A lot of the time, the tools can only provide a single piece of information that answers a specific question. It is obvious that multiple tools are needed to answer the questions posed by the participants in the paper. We found that both the questions participants posed and the tools they used to get the answers they needed were not aligned. Four ($C_{1,\ 3}$, CC_6, and B_3) of the six information needs (i.e., the most important, often requested, and time-consuming - $C_{1\text{-}3}$, $CC_{2,6}$, and B_3) can only be met manually by inspecting output from various tools, as shown in Table 4.2. Several companies have turned to Jenkins, GitHub, and Jira to meet these demands in the past. Practitioners, during interview in Paper I, stated that their organizations do not value these efforts (e.g., developing in-house solutions) and instead spend time investigating third-party tools or plugins, which results in increased manual work. In addition, practitioners may have difficulty deciding which tools to use to find answers to their information needs because of the wide variety of tools available.

Choosing the right tool to meet your information needs comes with a slew of difficulties. It is difficult to establish a tool that can meet the needs of multiple practitioners at the same time, because they have different information needs. Because each CI tool has its own preferences for input and output formats, it is more challenging to handle information needs because different CI logs or outputs may be required to answer them.

[1] https://www.merriam-webster.com/dictionary/trust

5.2 Diversity Based Testing: Adoption & Challenges - RQ 2 and RQ 4

Adoption of Test Optimization Techniques

Some needs can be met directly by aggregating data from the output of simple tools (i.e., loggers), while others necessitate the use of tools that incorporate sophisticated algorithms. For example, providing developers with faster feedback (Papers II and III) necessitated the use of tools that include test optimization techniques. There are many test optimization techniques in the literature, though they are not widely used in industry [99], [100]. Practitioners identified several reasons (Paper III) for not utilizing these well-established techniques, including: 1) a lack of explainable and comprehensible techniques, 2) a lack of testing process documentation within the organization, which further limits the use of techniques, 3) technical constraints and others (see Figure 4.14). Practitioners stated, during our investigation, that simple test optimization approaches are required that are suitable for usage in CI/CD setups without imposing excessive technical requirements. We plan to investigate the generalization of our findings in the future.

Diversity Based Testing is a Complement not a Replacement

Instead of replacing the overnight exhaustive test execution with DBT technique, we aim to provide developers with a preview of that execution, allowing them to prioritize code changes (i.e., which changes should be committed to the master branch) throughout their workday. Despite the fact that DBT provides comprehensive coverage even with smaller subsets, this does not guarantee that all defects will be discovered. The adoption of DBT assumes that the entire test suite is run overnight and failures should be reported the following day so that developers can debug and, if necessary, roll back their changes. Overall, the trade-off is to shorten feedback time while keeping sustainable coverage, while reducing the likelihood of dismissing test cases.

5.3 Flaky Tests - RQ 3 and RQ 4

Flaky Tests - The Dilemma and Human Factors

Since 2009, when Francis J. Lacoste [101] coined the term "flaky test", research publications have increased year after year, with a total of 24 studies published only in 2020 [102]. Researchers and software professionals are frustrated by the prevalence of flaky tests and are working to find reliable methods to reduce test flakiness. Research on flaky tests is mainly divided into causes and classification of flakiness [83]–[85], [103], detection of test flakiness [77], [87], [89], [104], and other relevant factors that affect test flakiness [90], [105], [106]. Different domains such as web applications, embedded systems, machine learning libraries, etc, have received attention from researcher to investigate the prevalence of test flakiness. Very few studies [28], [105], [107] have acually investigated test flakiness in real time industry with commercial software/hardware. In comparison with academic researcher, flaky test mitigation techniques and how practitioners deal with flaky tests in their day-to-day routines are of the utmost relevance.

When it comes to correcting test flakiness, human factors play a significant role [105] because developers and testers tend to blame each other for the flakiness in the testing process. If test flakiness is detected, software professionals are hesitant to accept responsibility. Due to a lack of understanding of the underlying causes of test flakiness, practitioners employ a "blame-others" strategy. According to testers, the flakiness is caused by changes in production code, however developers assert that test case codes are flaky. We suggest

that proper roles and responsibilities should be defined within the teams to detect and fix test flakiness. Another idea is to make use of the specifics about flakiness to encourage team members to take greater ownership of the flakiness problem. The flaky frequency, number of tests smells, and trace-back information, to name a few, can all be useful in resolving these kinds of conflicts. For example, using the information such as whether failed test case was executed on the latest code changes or not (e.g., trace-back), the number of tests smells in the test case, the flaky frequency can help in resolving such conflicts.

Flaky Tests - The Other Factors

The results interpretation after using machine learning classifiers to predict test flakiness is not simple. For example, looking only at the accuracy results (i.e., Figure 4.5) of classifiers can be deceiving. The important factor for classifier selection is to ask the right question and motivate the choice of using specific classifier such as **are we interested in detecting flaky tests correctly (i.e., precision) or marking a non flaky test as flaky is not cost effective (i.e, recall)**. It is important to look at precision, recall and accuracy all together for classifier selection. We can assume that practitioners are more interested in precision than recall because the test suite size, in many organizations, is very large and they cannot inspect all test cases. In this particular case, any classifier that correctly flag flaky tests will be encouraged. Precision can answer the question; ***"If the filter says this test case is flaky, what's the probability that it's flaky?"***.

In addition, the performance of machine learning classifiers depends on other factors such as: (1) whether the ML code is written by the non-programmer researcher, undertaking the research, using R libraries or some commercial/open-source libraries (Weka [108] in case of Pinto et. al [90]), (2) the selection of ML model based on the number of features in the training samples, (3) The variety provided by parameter tuning which might be problematic and necessitate unique considerations that can effect classifiers, etc.

5.4 Discussion Around Selected Methods - RQ 1 - RQ 4

Software engineering is a cross-disciplinary subject and it is important to understand the methods that are available to us, their limitations and when they can be applied. Basili [109] presented four different methods:

- Scientific: Observations are made of the world and a model is constructed, such as a simulation model.
- Engineering: Look at the current solutions, make some suggestions for improvement, and then assess the results.
- Analytical: After proposing a theoretical framework, it is compared to empirical evidence.
- Empirical: Empirical studies, such as case studies or experiments, are used to propose and evaluate a model.

The scientific method is employed in a variety of fields, such as simulating a telecommunications network to assess its performance [110]. The engineering method and the empirical method can be seen as variations of the scientific method [109]. Electromagnetic theory and algorithmic design are two examples where the analytical method has traditionally been applied [111]. Empirical studies are employed in social sciences and psychology, where we cannot state natural laws like in physics, thus heavily influenced by human behavior [111].

We considered empirical methods a suitable candidate due to the opportunity to work in collaboration with industry as well as lack of empirical evidence in software engineering

research [111]. Taking a step further, since we were dealing with practitioners and their challenges in achieving optimized CI/CD, we used mostly exploratory [112] methods except during Paper IV and V where we employed improving method. We did not employ descriptive methods because we were not interested in findings that described the ways in which the case study organizations operated – frequently by identifying roles or stakeholders and outlining a flow of activities. In addition, it is difficult to isolate specific factors and their causal relationships [67]when using descriptive methods. The reason to use exploratory method is that the objects are studied in their natural environment and findings are derived from the observations made.

Ethnographic investigations, such as workplace observations, could also yield valuable information about the research questions under consideration, particularly RQ1: identification of CI/CD information needs. A more accurate comparison between what practitioners say and what they actually do can be made by observing them in action as they go about their daily tasks at hand. Unfortunately, because to Covid-19 [2] constraints, we were not permitted to visit companies, which limited our methodological options.

5.5 Research & Practical Implications

During workshops, we demonstrated and presented our findings to practitioners through demos and presentations. The goal was to seek practitioner validation on whether or not our work may boost developer productivity in their day-to-day routine. We discovered that our work was well received because practitioners are struggling to address the complexity of CI/CD while providing faster feedback, increasing test/feature coverage, and decreasing test flakiness. Furthermore, we anticipate that our research will provide context for the development of tools and techniques in CI/CD to address a variety of challenges. For example, from the list of identified information needs in Paper I, practitioners might identify the most frequent and important needs that may be applicable in their CI/CD contexts and commence talks about systematic strategies to meet them. Throughout the investigation in Paper I, practitioners from various companies expressed similar needs, indicating repetition of information sought over time, necessitating the actions necessary to catalog information needs. Additionally, we provided a list of challenges experienced by practitioners when visualizing data in CI/CD, as well as recommendations for overcoming such challenges. We believe that practitioners will take these recommendations into account in their companies. Similarly, practitioners can benefit from our investigations into the use of test optimization techniques (e.g., DBT vs. other techniques) to achieve faster feedback and increased coverage, As a result of our research in the field of flaky test prevention, detection, and prediction, practitioners can not only detect flaky tests but also answer the question: *why is the test case flaky?* In order to resolve test flakiness, our work offers developers with an indication on where to investigate.

We believe that our research could serve as a foundation for additional work by other researchers interested in delving deeper into questions such as: what do identified information needs mean to different practitioners in different settings? Furthermore, researchers can conduct additional studies to gather information about what practitioners say and what they actually seek for. Validation of our challenges and recommendations (e.g., RQ 4), identified during a study in other contexts is greatly appreciated. The replication of our flaky test investigations will help to strengthen the community who will drive future research.

[2] https://www.who.int/emergencies/diseases/novel-coronavirus-2019

5.6 Future Work

The identification of information requirements was the result of collaboration between five software companies. It is essential, however, to investigate what practitioners state and what they do. An ethnographic study in a workplace context is one possibility for the near future. For example, a researcher might go to a site and collaborate with practitioners to observe how they go around gathering the answer they need. Such research will provide insights that may be useful in directing future work.

The tools and strategies presented in this work, we encourage practitioners to employ them in their day-to-day practice and share their experiences. It's worthwhile to dig deeper into the different scenarios in which tools and strategies have been evaluated. Evaluation in different context is important. We discovered that the flakiness of system testing in embedded system software may differ from that of stand-alone software that does not require any special hardware. The presence of flakiness in web applications may be different than in desktop programs. Research on test flakiness in embedded systems is scarce at the time of writing this dissertation. This is why we draw the attention of researchers and practitioners to this necessity. We concluded that test flakiness is context-dependent (i.e., open-source, closed-source, web programming, embedded systems, etc.).

6 Conclusion

Preliminary results suggest that practitioners have difficulties in adopting CI/CD and these difficulties grow when the practice requires optimization. The CI/CD optimization process can be approached in various ways, but our work focused on four: 1) understanding/identifying the information needs of software practitioners in CI/CD, 2) Adopting simple test optimization approaches that are realistic for use in contemporary CI/CD environments without adding too many technical requirements, 3) Understanding/identifying perceived causes and automated root cause analysis of test flakiness, thereby providing developers with indications on how to resolve test flakiness, leading to an increase in developer trust in feedback and 4) Understanding/identifying challenges in addressing information needs, providing faster feedback and trusting the feedback. We have conducted our investigation in close collaboration with embedded software industry in form of single or multiple case studies. We also used open-source data projects in some Papers for comparative analysis.

6.1 Information Needs - Answers to RQ1 and RQ 4

Testing, code and commit, confidence level, bug and artifacts were some of the 27 information needs that we identified for software professionals. There are many other software professionals besides developers who look for different kinds of information in their daily routines, such as managers, testers, and so on. There is a discrepancy between the identified needs and the techniques employed to meet them, according to our findings. Because some information needs cannot be met by current tools, manual inspections are required, which adds time to the process. Information about code & commit, confidence level, and testing is the most frequently sought for and most important information. Our research revealed eight challenges that practitioners encountered when creating or maintaining visualization tools. Our work compiled a list of ten recommendations from practitioners for those who are having difficulty visualizing information needs.

6.2 Test Optimization Techniques - Answers to RQ2 and RQ 4

Using a DBT approach to test case selection in continuous integration pipelines is the subject of our investigation. Integration level test cases given in natural language are compared using three different functions (Jaccard Index, Normalized Leven-Shtein, and Normalized Compression Distance). Our findings with two industrial partners show a significant reduction in the time (up to 92 percent faster) required to receive feedback from tests on a built version of the SUT. With a test suite cut by 85%, 65%, and 70%, we are able to cover all of the test requirements, dependencies, and processes completely and accurately. Furthermore, our findings show that the results of the diversity functions on different types of test artifacts differ and are comparable. We found that DBT can be used to increase the variety of a test set, thereby providing empirical evidence on the effectiveness of diversity based test optimization.

In addition to comparison of different functions within DBT, we examined three alternative methods of prioritization: diversity-based testing, which prioritizes the execution of a diverse set of test cases first, failure rate prioritization, which favors tests that failed in earlier builds, and time prioritization, which gives preference to the execution of tests that are executed quickly. Our findings show that diversity and failure-based prioritization are compatible, allowing testers to achieve high feature and failure coverage earlier in the testing cycle. As a result, time-based prioritization produces significantly faster test suites, but at the expense of very low feature and failure coverage. Thus, depending on their testing objective, we advise practitioners to prioritize diversity or failure rate.

To target stakeholders interested in implementing testing optimization approaches in CI/CD pipelines, our thematic study reveals 10 challenges and 7 reported effects. Our interview data reveals relevant process and practical aspects that foster adoption of these techniques. We concluded that developers want fast feedback on a large number of integrated features to verify their code changes, and they want their test suites to be linked to a confidence score that indicates the likelihood of missing failures due to fewer tests being run. These seven advantages are meant to encourage practitioners to use test prioritization, particularly diversity-based testing in CI pipelines, by highlighting the challenges and effects of such adoption. Additionally, our theme analysis suggests that many test infrastructure concerns, such as flaky tests and non-functional tests, are still unaddressed by test prioritization strategies. This finding suggests possible directions for future exploration.

6.3 Flaky Tests - Answers to RQ 3 and RQ 4

We presented the root causes of test flakiness experienced by professionals following a manual and automated analysis of the test cases code for flaky tests.We attempted to capture practitioners' perceptions of test flakiness and the factors that influence it. We discovered 19 perceived flakiness-increasing or decreasing factors. The identified perceived factors were categorized as Test Code, System Under Test, CI/Test Infrastructure, and Organization Related. Although the perceived factors aided the researched companies in reducing test flakiness, we concluded that what practitioners perceive as factors affecting test flakiness are, in reality, properties of a good test case (i.e., 'simple test case', 'small test case') and well defined software practices. Thus, Understanding the present perceptions or realities of practitioners is necessary to address test flakiness issues. For example, two of the identified perceived factors (i.e., test case size and simplicity) were observed in the test artifacts which represent that the practitioner's perceptions are important to consider to reduce test flakiness.

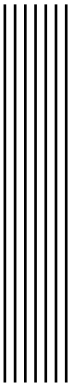
Detecting flaky tests and pinpointing the underlying causes take a significant amount of time for developers. We proposed a light weight technique named *trace-back coverage* to detect flaky tests. *Trace-back coverage* was combined with other factors (named as MDFlaker) such as test smells inducing test flakiness, flaky frequency and test case size to investigate the effect on revealing test flakiness. When all factors are taken into consideration, the result's accuracy is increased. There are many advantages to MDFlaker over other tools, including less work to implement for any language, shorter execution times and more information for root cause analysis.

Different sophisticated machine learning algorithms are being used in our efforts to address flaky test concerns. Open-source data was used as a testing ground for three machine learning classifiers: Naive Bayes (NB), Support Vector Machine (SVM), and Random Forest (RF). In comparison to NBL (precision 70% and recall $>$ 30%) and SVM (precision 70% and recall $>$ 60%), we found that only RF had higher accuracy (i.e., $>$ 90%), but recall was very low (i.e., 10%). We concluded that predicting accuracy of ML classifiers are strongly associated with the lexical information of test cases (i.e., test cases written in `Java` or `Python`). We investigated why other classifiers failed to produce expected results and concluded that; 1) flaky test cases are caused by a combination of the test smell and an external environment; however, the external environment was not considered in this study, 2) Classifiers using machine learning should take into account not only the frequency of test smells in the test case, but also other significant test case syntax that have the power to cancel out the effect of test smells.

Papers

The papers associated with this thesis have been removed for copyright reasons. For more details about these see:

https://doi.org/10.3384/9789179294236



Bibliography

[1] Jez Humble and David Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, en. Addison Wesley, 2011. [Online]. Available: https://www.pearson.com/content/one-dot-com/one-dot-com/us/en/higher-education/program.html (visited on 12/28/2021).

[2] F. Zampetti, S. Scalabrino, R. Oliveto, G. Canfora, and M. Di Penta, "How Open Source Projects Use Static Code Analysis Tools in Continuous Integration Pipelines," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, May 2017, pp. 334–344. DOI: 10.1109/MSR.2017.2.

[3] J. Gmeiner, R. Ramler, and J. Haslinger, "Automated testing in the continuous delivery pipeline: A case study of an online company," in *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Apr. 2015, pp. 1–6. DOI: 10.1109/ICSTW.2015.7107423.

[4] J. Wettinger, U. Breitenbücher, M. Falkenthal, and F. Leymann, "Collaborative gathering and continuous delivery of DevOps solutions through repositories," en, *Computer Science - Research and Development*, vol. 32, no. 3, pp. 281–290, Jul. 2017, ISSN: 1865-2042. DOI: 10.1007/s00450-016-0338-z. [Online]. Available: https://doi.org/10.1007/s00450-016-0338-z (visited on 12/28/2021).

[5] C. Vassallo, F. Zampetti, D. Romano, M. Beller, A. Panichella, M. D. Penta, and A. Zaidman, "Continuous Delivery Practices in a Large

Financial Organization," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Oct. 2016, pp. 519–528. DOI: 10.1109/ICSME.2016.72.

[6] P. Rodríguez, A. Haghighatkhah, L. E. Lwakatare, S. Teppola, T. Suomalainen, J. Eskeli, T. Karvonen, P. Kuvaja, J. M. Verner, and M. Oivo, "Continuous deployment of software intensive products and services: A systematic mapping study," *Journal of Systems and Software*, vol. 123, pp. 263–291, Jan. 2017, ISSN: 0164-1212. DOI: 10.1016/j.jss.2015.12.015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121215002812.

[7] M. Leppänen, S. Mäkinen, M. Pagels, V. P. Eloranta, J. Itkonen, M. V. Mäntylä, and T. Männistö, "The highways and country roads to continuous deployment," *IEEE Software*, vol. 32, no. 2, pp. 64–72, Mar. 2015, ISSN: 0740-7459. DOI: 10.1109/MS.2015.50.

[8] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig, "Usage, costs, and benefits of continuous integration in open-source projects," in *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Sep. 2016, pp. 426–437.

[9] D. Ståhl and J. Bosch, "Experienced benefits of continuous integration in industry software product development: A case study," in *The 12th IASTED International Conference on Software Engineering*, 2013, pp. 736–743.

[10] D. Ståhl, K. Hallén, and J. Bosch, "Achieving traceability in large scale continuous integration and delivery deployment, usage and validation of the eiffel framework," en, *Empirical Software Engineering*, vol. 22, no. 3, pp. 967–995, Jun. 2017, ISSN: 1573-7616. DOI: 10.1007/s10664-016-9457-1. [Online]. Available: https://doi.org/10.1007/s10664-016-9457-1 (visited on 03/06/2020).

[11] J. Josyula, S. Panamgipalli, M. Usman, R. Britto, and N. Bin Ali, "Software Practitioners' Information Needs and Sources: A Survey Study," in *2018 9th International Workshop on Empirical Software Engineering in Practice (IWESEP)*, ISSN: 2333-519X, Dec. 2018, pp. 1–6. DOI: 10.1109/IWESEP.2018.00009.

[12] A. J. Ko, R. DeLine, and G. Venolia, "Information Needs in Collocated Software Development Teams," in *Proceedings of the 29th international conference on Software Engineering*, ser. ICSE '07, USA: IEEE Computer Society, May 2007, pp. 344–353, ISBN: 978-0-7695-2828-1. DOI: 10.1109/ICSE.2007.45. [Online]. Available: https://doi.org/10.1109/ICSE.2007.45 (visited on 03/04/2020).

[13] T. D. LaToza and B. A. Myers, "Developers ask reachability questions," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10, Cape Town, South Africa: Association for Computing Machinery, May 2010, pp. 185–194, ISBN: 978-1-60558-719-6. DOI: 10.1145/1806799.1806829. [Online]. Available: https://doi.org/10.1145/1806799.1806829 (visited on 03/04/2020).

[14] J. Sillito, G. C. Murphy, and K. De Volder, "Questions programmers ask during software evolution tasks," in *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, ser. SIGSOFT '06/FSE-14, Portland, Oregon, USA: Association for Computing Machinery, Nov. 2006, pp. 23–34, ISBN: 978-1-59593-468-0. DOI: 10.1145/1181775.1181779. [Online]. Available: https://doi.org/10.1145/1181775.1181779 (visited on 03/04/2020).

[15] A. Memon, Z. Gao, B. Nguyen, S. Dhanda, E. Nickell, R. Siemborski, and J. Micco, "Taming google-scale continuous testing," in *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, IEEE, 2017, pp. 233–242.

[16] B. Miranda, E. Cruciani, R. Verdecchia, and A. Bertolino, "FAST approaches to scalable similarity-based test case prioritization," in *Proceedings of the 40th International Conference on Software Engineering*, ser. ICSE '18, Gothenburg, Sweden: ACM, 2018, pp. 222–232, ISBN: 978-1-4503-5638-1. DOI: 10.1145/3180155.3180210.

[17] F. G. de Oliveira Neto, A. Ahmad, O. Leifler, K. Sandahl, and E. Enoiu, "Improving continuous integration with similarity-based test case selection," in *Proceedings of the 13th International Workshop on Automation of Software Test*, Gothenburg, Sweden: ACM, 2018, pp. 39–45, ISBN: 978-1-4503-5743-2. DOI: 10.1145/3194733.3194744.

[18] P. E. Strandberg, E. P. Enoiu, W. Afzal, D. Sundmark, and R. Feldt, "Information flow in software testing–an interview study with embedded software engineering practitioners," *IEEE Access*, vol. 7, pp. 46 434–46 453, 2019.

[19] F. G. de Oliveira Neto, R. Feldt, L. Erlenhov, and J. B. d. S. Nunes, "Visualizing test diversity to support test optimisation," in *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, Dec. 2018, pp. 149–158.

[20] S. Tahvili, L. Hatvani, M. Felderer, W. Afzal, and M. Bohlin, "Automated functional dependency detection between test cases using doc2vec and clustering," in *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*, 2019, pp. 19–26.

[21] M. Shahin, M. A. Babar, and L. Zhu, "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices," *IEEE Access*, vol. 5, pp. 3909–3943, 2017. DOI: 10.1109/ACCESS.2017.2685629.

[22] J. Anderson, S. Salem, and H. Do, "Improving the Effectiveness of Test Suite Through Mining Historical Data," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014, New York, NY, USA: ACM, 2014, pp. 142–151, ISBN: 978-1-4503-2863-0. DOI: 10.1145/2597073.2597084. [Online]. Available: http://doi.acm.org/10.1145/2597073.2597084.

[23] M. Leppänen, S. Mäkinen, M. Pagels, V. P. Eloranta, J. Itkonen, M. V. Mäntylä, and T. Männistö, "The highways and country roads to continuous deployment," *IEEE Software*, vol. 32, no. 2, pp. 64–72, Mar. 2015, ISSN: 0740-7459. DOI: 10.1109/MS.2015.50.

[24] O. McHugh, K. Conboy, and M. Lang, "Agile Practices: The Impact on Trust in Software Project Teams," *IEEE Software*, vol. 29, no. 3, pp. 71–76, May 2012, Conference Name: IEEE Software, ISSN: 1937-4194. DOI: 10.1109/MS.2011.118.

[25] Q. Luo, F. Hariri, L. Eloussi, and D. Marinov, "An empirical analysis of flaky tests," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2014, Hong Kong, China: Association for Computing Machinery, 2014, pp. 643–653, ISBN: 9781450330565. DOI: 10.1145/2635868.2635920. [Online]. Available: https://doi.org/10.1145/2635868.2635920.

[26] W. Lam, R. Oei, A. Shi, D. Marinov, and T. Xie, "Idflakies: A framework for detecting and partially classifying flaky tests," in *2019 12th ieee conference on software testing, validation and verification (icst)*, IEEE, 2019, pp. 312–322.

[27] M. Fowler, *Eradicating Non-Determinism in Tests. Accessed [2019-04-15 18:52:30]*. [Online]. Available: https://martinfowler.com/articles/nonDeterminism.html (visited on 04/15/2019).

[28] J. Bell, O. Legunsen, M. Hilton, L. Eloussi, T. Yung, and D. Marinov, "Deflaker: Automatically detecting flaky tests," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, IEEE, 2018, pp. 433–444.

[29] C. Leong, A. Singh, M. Papadakis, Y. L. Traon, and J. Micco, "Assessing transition-based test selection algorithms at Google," in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*, ser. ICSE-SEIP '19, Montreal, Quebec, Canada: IEEE Press, May 2019, pp. 101–110. DOI: 10.1109/ICSE-SEIP.2019.00019. [Online]. Available: https://doi.org/10.1109/ICSE-SEIP.2019.00019 (visited on 02/17/2021).

[30] J. Micco, *Flaky Tests at Google and How We Mitigate Them, https://testing.googleblog.com/2016/05/flaky-tests-at-google-and-how-we.html. Accessed [2019-04-15 18:48:16]*, en. [Online]. Available: https://testing.googleblog.com/2016/05/flaky-tests-at-google-and-how-we.html (visited on 04/15/2019).

[31] "IEEE Standard Glossary of Software Engineering Terminology," *IEEE Std 610.12-1990*, pp. 1–84, Dec. 1990, Conference Name: IEEE Std 610.12-1990. DOI: 10.1109/IEEESTD.1990.101064.

[32] M. S. Feather, T. Menzies, and J. R. Connelly, "Matching Software Practitioner Needs to Researcher Activities," in *Proceedings of the Tenth Asia-Pacific Software Engineering Conference Software Engineering Conference*, ser. APSEC '03, USA: IEEE Computer Society, Dec. 2003, p. 6, ISBN: 978-0-7695-2011-7. (visited on 03/04/2020).

[33] P. E. Strandberg, W. Afzal, and D. Sundmark, "Software test results exploration and visualization with continuous integration and nightly testing," en, *International Journal on Software Tools for Technology Transfer*, vol. 24, no. 2, pp. 261–285, Apr. 2022, ISSN: 1433-2787. DOI: 10.1007/s10009-022-00647-1. [Online]. Available: https://doi.org/10.1007/s10009-022-00647-1 (visited on 06/20/2022).

[34] C. Persson and N. Yilmazturk, "Establishment of automated regression testing at ABB: Industrial experience report on 'avoiding the pitfalls'," in *Proceedings. 19th International Conference on Automated Software Engineering, 2004.*, ISSN: 1938-4300, Sep. 2004, pp. 112–121. DOI: 10.1109/ASE.2004.1342729.

[35] E. Engström and P. Runeson, "A Qualitative Survey of Regression Testing Practices," en, in *Product-Focused Software Process Improvement*, M. Ali Babar, M. Vierimaa, and M. Oivo, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2010, pp. 3–16, ISBN: 978-3-642-13792-1. DOI: 10.1007/978-3-642-13792-1_3.

[36] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software Testing Verification and Reliability*, vol. 22, no. 2, pp. 67–120, 2012, ISSN: 10991689. DOI: 10.1002/stvr.430.

[37] C. Henard, M. Papadakis, M. Harman, Y. Jia, and Y. Le Traon, "Comparing white-box and black-box test prioritization," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, May 2016, pp. 523–534. DOI: 10.1145/2884781.2884791.

[38] E. G. Cartaxo, P. D. L. Machado, and F. G. de Oliveira Neto, "On the use of a similarity function for test case selection in the context of model-based testing," *Software Testing, Verification and Reliability*, vol. 21, no. 2, pp. 75–100, 2011, ISSN: 1099-1689. DOI: 10.1002/stvr.413.

[39] H. Hemmati, A. Arcuri, and L. Briand, "Achieving scalable model-based testing through test case diversity," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 1, 6:1–6:42, Mar. 2013, ISSN: 1049-331X. DOI: 10.1145/2430536.2430540.

[40] R. Feldt, S. Poulding, D. Clark, and S. Yoo, "Test set diameter: Quantifying the diversity of sets of test cases," in *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, Apr. 2016, pp. 223–233. DOI: 10.1109/ICST.2016.33.

[41] F. G. de Oliveira Neto, F. Dobslaw, and R. Feldt, "Using mutation testing to measure behavioural test diversity," in *13th IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Mar. 2020, pp. 254–263. DOI: 10.1109/ICSTW50294.2020.00051.

[42] T. B. Noor and H. Hemmati, "A similarity-based approach for test case prioritization using historical failure data," in *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, Nov. 2015, pp. 58–68. DOI: 10.1109/ISSRE.2015.7381799.

[43] A. Rainer, T. Hall, and N. Baddoo, "Persuading Developers to 'Buy into' Software Process Improvement: Local Opinion and Empirical Evidence," in *Proceedings of the 2003 International Symposium on Empirical Software Engineering*, ser. ISESE '03, Washington, DC, USA: IEEE Computer Society, 2003, pp. 326–, ISBN: 978-0-7695-2002-5. [Online]. Available: http://dl.acm.org/citation.cfm?id=942801.943629 (visited on 04/17/2019).

[44] E. Laukkanen, M. Paasivaara, and T. Arvonen, "Stakeholder Perceptions of the Adoption of Continuous Integration – A Case Study," in *2015 Agile Conference*, Aug. 2015, pp. 11–20. DOI: 10.1109/Agile.2015.15.

[45] W. Sun, G. Marakas, and M. Aguirre-Urreta, "The Effectiveness of Pair Programming: Software Professionals' Perceptions," *IEEE Software*, vol. 33, no. 4, pp. 72–79, Jul. 2016, ISSN: 0740-7459. DOI: 10.1109/MS.2015.106.

[46] F. Ebert and F. Castor, "A Study on Developers' Perceptions about Exception Handling Bugs," in *2013 IEEE International Conference on Software Maintenance*, Sep. 2013, pp. 448–451. DOI: 10.1109/ICSM.2013.69.

[47] H. Shah, C. Gorg, and M. J. Harrold, "Understanding Exception Handling: Viewpoints of Novices and Experts," *IEEE Transactions on Software Engineering*, vol. 36, no. 2, pp. 150–161, Mar. 2010, ISSN: 0098-5589. DOI: 10.1109/TSE.2010.7.

[48] F. Palomba, G. Bavota, M. D. Penta, R. Oliveto, and A. D. Lucia, "Do They Really Smell Bad? A Study on Developers' Perception of Bad Code Smells," in *2014 IEEE International Conference on Software Maintenance and Evolution*, Sep. 2014, pp. 101–110. DOI: 10.1109/ICSME.2014.32.

[49] C. R. Camacho, S. Marczak, and D. S. Cruzes, "Agile Team Members Perceptions on Non-functional Testing: Influencing Factors from an Empirical Study," in *2016 11th International Conference on Availability, Reliability and Security (ARES)*, Aug. 2016, pp. 582–589. DOI: 10.1109/ARES.2016.98.

[50] J. Percival and N. Harrison, "Developer Perceptions of Process Desirability: Test Driven Development and Cleanroom Compared," in *2013 46th Hawaii International Conference on System Sciences*, Jan. 2013, pp. 4800–4809. DOI: 10.1109/HICSS.2013.175.

[51] H. Tan and V. Tarasov, "Test Case Quality as Perceived in Sweden," in *2018 IEEE/ACM 5th International Workshop on Requirements Engineering and Testing (RET)*, Jun. 2018, pp. 9–12.

[52] G. Bavota, A. Qusef, R. Oliveto, A. De Lucia, and D. Binkley, "Are test smells really harmful? An empirical study," en, *Empirical Software Engineering*, vol. 20, no. 4, pp. 1052–1094, Aug. 2015, ISSN: 1573-7616. DOI: 10.1007/s10664-014-9313-0. [Online]. Available: https://doi.org/10.1007/s10664-014-9313-0 (visited on 05/22/2019).

[53] Z. Wan, X. Xia, A. E. Hassan, D. Lo, J. Yin, and X. Yang, "Perceptions, Expectations, and Challenges in Defect Prediction," *IEEE Transactions on Software Engineering*, pp. 1–1, 2018, ISSN: 0098-5589. DOI: 10.1109/TSE.2018.2877678.

[54] W. Zou, D. Lo, Z. Chen, X. Xia, Y. Feng, and B. Xu, "How Practitioners Perceive Automated Bug Report Management Techniques," *IEEE Transactions on Software Engineering*, pp. 1–1, 2018, ISSN: 0098-5589. DOI: 10.1109/TSE.2018.2870414.

[55] Z. S. H. Abad, G. Ruhe, and M. Bauer, "Task Interruptions in Requirements Engineering: Reality Versus Perceptions!" In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, ISSN: 2332-6441, Sep. 2017, pp. 342–351. DOI: 10.1109/RE.2017.75.

[56] D. Silva, L. Teixeira, and M. d'Amorim, "Shake It! Detecting Flaky Tests Caused by Concurrency with Shaker," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, ISSN: 2576-3148, Sep. 2020, pp. 301–311. DOI: 10.1109/ICSME46990.2020.00037.

[57] A. Haghighatkhah, M. Mäntylä, M. Oivo, and P. Kuvaja, "Test prioritization in continuous integration environments," *Journal of Systems and Software*, vol. 146, pp. 80–98, 2018, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2018.08.061. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121218301730.

[58] H. Spieker, A. Gotlieb, D. Marijan, and M. Mossige, "Reinforcement learning for automatic test case prioritization and selection in continuous integration," in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2017, Santa Barbara, CA, USA: Association for Computing Machinery, 2017, pp. 12–22, ISBN: 9781450350761. DOI: 10.1145/3092703.3092709. [Online]. Available: https://doi.org/10.1145/3092703.3092709.

[59] S. Elbaum, G. Rothermel, and J. Penix, "Techniques for improving regression testing in continuous integration development environments," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2014, Hong Kong, China: Association for Computing Machinery, 2014, pp. 235–245, ISBN: 9781450330565. DOI: 10.1145/2635868.2635910. [Online]. Available: https://doi.org/10.1145/2635868.2635910.

[60] F. G. de Oliveira Neto, R. Torkar, R. Feldt, L. Gren, C. A. Furia, and Z. Huang, "Evolution of statistical analysis in empirical software engineering research: Current state and steps forward," *Journal of Systems and Software*, vol. 156, pp. 246–267, 2019, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2019.07.002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121219301451.

[61] E. Engström, K. Petersen, N. B. Ali, and E. Bjarnason, "Serp-test: A taxonomy for supporting industry–academia communication," eng, *Software Quality Journal*, vol. 25, no. 4, pp. 1269–1305, 2017, ISSN: 0963-9314. DOI: 10.1007/s11219-016-9322-x. [Online]. Available: http://dx.doi.org/10.1007/s11219-016-9322-x.

[62] N. bin Ali, E. Engström, M. Taromirad, M. R. Mousavi, N. M. Minhas, D. Helgesson, S. Kunze, and M. Varshosaz, "On the search for industry-relevant regression testing research," *Empirical Software Engineering*, vol. 24, no. 4, pp. 2020–2055, 2019.

[63] M. P. Cota, M. D. Rodríguez, M. R. González-Castro, and R. M. M. Gonçalves, "Massive data visualization analysis analysis of current visualization techniques and main challenges for the future," in *2017 12th Iberian Conference on Information Systems and Technologies (CISTI)*, Jun. 2017, pp. 1–6. DOI: 10.23919/CISTI.2017.7975704.

[64] G. S. D. Wedawatta, M. J. B. Ingirige, and R. D. G. Amaratunga, "Case study as a research strategy: Investigating extreme weather resilience of construction SMEs in the UK," en, Kandalama, Sri Lanka, Jul. 2011. [Online]. Available: http://usir.salford.ac.uk/id/eprint/18250/ (visited on 01/06/2022).

[65] A. OPOKU, V. AHMED, and J. AKOTIA, "Choosing an appropriate research methodology and method," in *Research Methodology in the Built Environment*, Num Pages: 19, Routledge, 2016, ISBN: 978-1-315-72552-9.

[66] S. Bailey and D. Handu, *Introduction to Epidemiologic Research Methods in Public Health Practice*, English, 1st edition. Burlington, MA: Jones & Bartlett Learning, Aug. 2012, ISBN: 978-1-4496-2784-3.

[67] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, p. 131, Dec. 2008, ISSN: 1573-7616. DOI: 10.1007/s10664-008-9102-8. [Online]. Available: https://doi.org/10.1007/s10664-008-9102-8.

[68] C. Robson, *Real World Research: A Resource for Social Scientists and Practitioner-Researchers*, English, 2nd edition. Oxford, UK ; Madden, Mass: Wiley-Blackwell, Mar. 2002, ISBN: 978-0-631-21305-5.

[69] R. K. Yin, *Case study research design and methods*, 4th ed. Thousand Oaks, Calif Sage Publications, 2009, ISBN: 978-1-4129-6099-1. [Online]. Available: https://trove.nla.gov.au/work/11329910 (visited on 04/17/2019).

[70] J. W. Creswell, *Research design: qualitative, quantitative, and mixed methods approaches*, 4th ed. Thousand Oaks, California: SAGE Publications, 2014, ISBN: 978-1-4522-2609-5 978-1-4522-2610-1.

[71] M. Q. Patton, "Evaluation in the Field: The Need for Site Visit Standards," en, *American Journal of Evaluation*, vol. 36, no. 4, pp. 444–460, Dec. 2015, Publisher: SAGE Publications Inc, ISSN: 1098-2140. DOI: 10.1177/1098214015600785. [Online]. Available: https://doi.org/10.1177/1098214015600785 (visited on 01/17/2022).

[72] K. Thoring, R. Mueller, and P. Badke-Schaub, *Workshops as a Research Method: Guidelines for Designing and Evaluating Artifacts Through Workshops*, eng. Jan. 2020, ISBN: 978-0-9981331-3-3. [Online]. Available: http://hdl.handle.net/10125/64362 (visited on 06/29/2022).

[73] M. Morisio, I. Stamelos, and A. Tsoukias, "A new method to evaluate software artifacts against predefined profiles," in *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, ser. SEKE '02, New York, NY, USA: Association for Computing Machinery, Jul. 2002, pp. 811–818, ISBN: 978-1-58113-556-5. DOI: 10.1145/568760.568899. [Online]. Available: http://doi.org/10.1145/568760.568899 (visited on 01/17/2022).

[74] R. E. Stake, *The Art of Case Study Research*, en. SAGE, Apr. 1995, Google-Books-ID: ApGdBx76b9kC, ISBN: 978-0-8039-5767-1.

[75] A. Strauss and J. Corbin, *Basics of qualitative research: Techniques and procedures for developing grounded theory, 2nd ed.* Thousand Oaks, CA, US: Sage Publications, Inc, 1998, ISBN: 978-0-8039-5939-2 978-0-8039-5940-8.

[76] V. Braun, V. Clarke, and G. Terry, "Thematic analysis," *APA handbook of research methods in psychology*, vol. 2, pp. 57–71, 2012.

[77] W. Lam, R. Oei, A. Shi, D. Marinov, and T. Xie, "iDFlakies: A Framework for Detecting and Partially Classifying Flaky Tests," in *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, ISSN: 2159-4848, Apr. 2019, pp. 312–322. DOI: 10.1109/ICST.2019.00038.

[78] M. Sasaki and H. Shinnou, "Spam detection using text clustering," in *2005 International Conference on Cyberworlds (CW'05)*, ISSN: null, Nov. 2005, 4 pp.–319. DOI: 10.1109/CW.2005.83.

[79] M. R. Islam, W. Zhou, and M. U. Choudhury, "Dynamic Feature Selection for Spam Filtering Using Support Vector Machine," in *6th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007)*, ISSN: null, Jul. 2007, pp. 757–762. DOI: 10.1109/ICIS.2007.92.

[80] E. G. Dada, J. S. Bassi, H. Chiroma, S. M. Abdulhamid, A. O. Adetunmbi, and O. E. Ajibuwa, "Machine learning for email spam filtering: Review, approaches and open research problems," en, *Heliyon*, vol. 5, no. 6, e01802, Jun. 2019, ISSN: 2405-8440. DOI: 10.1016/j.heliyon.2019.e01802. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2405844018353404 (visited on 01/20/2020).

[81] *Jenkinsci/eiffel-broadcaster-plugin*, original-date: 2019-01-22T15:04:13Z, Jul. 2019. [Online]. Available: https://github.com/jenkinsci/eiffel-broadcaster-plugin (visited on 03/27/2020).

[82] H. Hemmati, Z. Fang, and M. V. Mantyla, "Prioritizing Manual Test Cases in Traditional and Rapid Release Environments," in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, Apr. 2015, pp. 1–10. DOI: 10.1109/ICST.2015.7102602.

[83] Q. Luo, F. Hariri, L. Eloussi, and D. Marinov, "An Empirical Analysis of Flaky Tests," in *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2014, event-place: Hong Kong, China, New York, NY, USA: ACM, 2014, pp. 643–653, ISBN: 978-1-4503-3056-5. DOI: 10.1145/2635868.2635920. [Online]. Available: http://doi.acm.org/10.1145/2635868.2635920 (visited on 04/15/2019).

[84] S. Thorve, C. Sreshtha, and N. Meng, "An Empirical Study of Flaky Tests in Android Apps," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Sep. 2018, pp. 534–538. DOI: 10.1109/ICSME.2018.00062.

[85] M. Eck, F. Palomba, M. Castelluccio, and A. Bacchelli, "Understanding Flaky Tests: The Developer's Perspective," *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2019*, pp. 830–840, 2019, arXiv: 1907.01466. DOI: 10.1145/3338906.3338945. [Online]. Available: http://arxiv.org/abs/1907.01466 (visited on 08/15/2019).

[86] A. Sjöbom, *Studying Test Flakiness in Python Projects : Original Findings for Machine Learning*, eng. 2019. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-264459 (visited on 01/20/2020).

[87] J. Bell, O. Legunsen, M. Hilton, L. Eloussi, T. Yung, and D. Marinov, "DeFlaker: Automatically Detecting Flaky Tests," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, May 2018, pp. 433–444. DOI: 10.1145/3180155.3180164.

[88] S. Liviu, *A machine learning solution for detecting and mitigating flaky tests*, en, Oct. 2019. [Online]. Available: https://medium.com/fitbit-tech-blog/a-machine-learning-solution-for-detecting-and-mitigating-flaky-tests-c5626ca7e853 (visited on 12/29/2020).

[89] T. M. King, D. Santiago, J. Phillips, and P. J. Clarke, "Towards a Bayesian Network Model for Predicting Flaky Automated Tests," in *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, Lisbon: IEEE Comput. Soc, Jul. 2018, pp. 100–107. DOI: 10.1109/QRS-C.2018.00031.

[90] G. Pinto, B. Miranda, S. Dissanayake, M. d'Amorim, C. Treude, and A. Bertolino, "What is the Vocabulary of Flaky Tests?" In *Proceedings of the 17th International Conference on Mining Software Repositories*, New York, NY, USA: Association for Computing Machinery, Jun. 2020, pp. 492–502, ISBN: 978-1-4503-7517-7. [Online]. Available: http://doi.org/10.1145/3379597.3387482 (visited on 11/26/2020).

[91] T. Fawcett, "An introduction to ROC analysis," en, *Pattern Recognition Letters*, ROC Analysis in Pattern Recognition, vol. 27, no. 8, pp. 861–874, Jun. 2006, ISSN: 0167-8655. DOI: 10.1016/j.patrec.2005.10.010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016786550500303X (visited on 01/16/2020).

[92] D. Zhang, J. Wang, and X. Zhao, "Estimating the Uncertainty of Average F1 Scores," in *Proceedings of the 2015 International Conference on The Theory of Information Retrieval*, ser. ICTIR '15, Northampton, Massachusetts, USA: Association for Computing Machinery, Sep. 2015, pp. 317–320, ISBN: 978-1-4503-3833-2. DOI: 10.1145/2808194.2809488. [Online]. Available: https://doi.org/10.1145/2808194.2809488 (visited on 01/21/2020).

[93] Wang, *Baselines and bigrams | Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2*. [Online]. Available: https://dl-acm-org.e.bibl.liu.se/doi/10.5555/2390665.2390688 (visited on 01/25/2020).

[94] S. Abu-Nimeh, D. Nappa, X. Wang, and S. Nair, "A comparison of machine learning techniques for phishing detection," en, in *Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit on - eCrime '07*, Pittsburgh, Pennsylvania: ACM Press, 2007, pp. 60–69. DOI: 10.1145/1299015.1299021. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1299015.1299021 (visited on 01/20/2020).

[95] L. Breiman, "Random Forests," en, *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001, ISSN: 1573-0565. DOI: 10.1023/A:1010933404324. [Online]. Available: https://doi.org/10.1023/A:1010933404324 (visited on 01/14/2020).

[96] F. Palomba and A. Zaidman, "The smell of fear: On the relation between test smells and flaky tests," en, *Empirical Software Engineering*, vol. 24, no. 5, pp. 2907–2946, Oct. 2019, ISSN: 1573-7616. DOI: 10.1007/s10664-019-09683-z. [Online]. Available: https://doi.org/10.1007/s10664-019-09683-z (visited on 08/23/2019).

[97] A. Vance, C. Elie-Dit-Cosaque, and D. Straub, "Examining Trust in Information Technology Artifacts: The Effects of System Quality and Culture," *Journal of Management Information Systems*, vol. 24, no. 4, pp. 73–100, Apr. 2008, ISSN: 0742-1222. DOI: 10.2753/MIS0742-

1222240403. [Online]. Available: http://doi.org/10.2753/MIS0742-1222240403 (visited on 05/06/2022).

[98] J. Krüger, W. Gu, H. Shen, M. Mukelabai, R. Hebig, and T. Berger, "Towards a Better Understanding of Software Features and Their Characteristics: A Case Study of Marlin," in *Proceedings of the 12th International Workshop on Variability Modelling of Software-Intensive Systems*, ser. VAMOS 2018, New York, NY, USA: Association for Computing Machinery, Feb. 2018, pp. 105–112, ISBN: 978-1-4503-5398-4. DOI: 10.1145/3168365.3168371. [Online]. Available: http://doi.org/10.1145/3168365.3168371 (visited on 05/06/2022).

[99] N. Gupta, A. Sharma, and M. K. Pachariya, "An Insight Into Test Case Optimization: Ideas and Trends With Future Perspectives," *IEEE Access*, vol. 7, pp. 22 310–22 327, 2019, Conference Name: IEEE Access, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2899471.

[100] K. Doganay, "Applications of Optimization Methods in Industrial Maintenance Scheduling and Software Testing," eng, 2014, Publisher: Mälardalen University. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:mdh:diva-25944 (visited on 03/10/2022).

[101] F. J. Lacoste, "Killing the Gatekeeper: Introducing a Continuous Integration System," in *2009 Agile Conference*, Aug. 2009, pp. 387–392. DOI: 10.1109/AGILE.2009.35.

[102] O. Parry, G. M. Kapfhammer, M. Hilton, and P. McMinn, "A Survey of Flaky Tests," *ACM Transactions on Software Engineering and Methodology*, vol. 31, no. 1, 17:1–17:74, Oct. 2021, ISSN: 1049-331X. DOI: 10.1145/3476105. [Online]. Available: http://doi.org/10.1145/3476105 (visited on 03/08/2022).

[103] W. Lam, P. Godefroid, S. Nath, A. Santhiar, and S. Thummalapenta, "Root causing flaky tests in a large-scale industrial setting," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2019, New York, NY, USA: Association for Computing Machinery, Jul. 2019, pp. 101–111, ISBN: 978-1-4503-6224-5. DOI: 10.1145/3293882.3330570. [Online]. Available: https://doi.org/10.1145/3293882.3330570 (visited on 09/08/2020).

[104] A. Bertolino, B. Miranda, and R. Verdecchia, "Know your neighbor: Fast static prediction of test flakiness," en, *ISTI Technical Reports*, vol. 2020, no. 001, p. 1, Jan. 2020. DOI: 10.32079/ISTI-TR-2020/001. [Online]. Available: https://doi.org/10.32079/ISTI-TR-2020/001 (visited on 12/31/2020).

[105] A. Ahmad, O. Leifler, and K. Sandahl, "Empirical Analysis of Factors and their Effect on Test Flakiness - Practitioners' Perceptions," *arXiv:1906.00673 [cs]*, Jun. 2019, arXiv: 1906.00673. [Online]. Available: http://arxiv.org/abs/1906.00673 (visited on 01/20/2020).

[106] A. Ahmad, O. Leifler, and K. Sandahl, "An Evaluation of Machine Learning Methods for Predicting Flaky Tests," en, in *8th International Workshop on Quantitative Approaches to Software Quality in conjunction with the 27th Asia-Pacifc SoftwareEngineering Conference (APSEC 2020) Singapore*, Dec. 2020, pp. 37–46.

[107] W. Lam, K. Muşlu, H. Sajnani, and S. Thummalapenta, "A study on the lifecycle of flaky tests," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20, New York, NY, USA: Association for Computing Machinery, Jun. 2020, pp. 1471–1482, ISBN: 978-1-4503-7121-6. DOI: 10.1145/3377811.3381749. [Online]. Available: https://doi.org/10.1145/3377811.3381749 (visited on 04/30/2021).

[108] I. H. Witten and E. Frank, "Data mining: Practical machine learning tools and techniques with Java implementations," *ACM SIGMOD Record*, vol. 31, no. 1, pp. 76–77, Mar. 2002, ISSN: 0163-5808. DOI: 10.1145/507338.507355. [Online]. Available: https://doi.org/10.1145/507338.507355 (visited on 10/09/2020).

[109] "Basili, V.R.: The experimental paradigm in software engineering. In: H.D. Rombach, V.R. Basili, R.W. Selby (eds.) Experimental Software Engineering Issues: Critical Assessment and Future Directives. Lecture Notes in Computer Science, vol. 706. Springer, Berlin Heidelberg (1993)."

[110] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, "Analysis and Interpretation," en, in *Experimentation in Software Engineering*, C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, Eds., Berlin, Heidelberg: Springer, 2012, pp. 123–151, ISBN: 978-3-642-29044-2. DOI: 10.1007/978-3-642-29044-2_10. [Online]. Available: https://doi.org/10.1007/978-3-642-29044-2_10 (visited on 03/09/2022).

[111] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*, en. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ISBN: 978-3-642-29043-5 978-3-642-29044-2. DOI: 10.1007/978-3-642-29044-2. [Online]. Available: http://link.springer.com/10.1007/978-3-642-29044-2 (visited on 03/09/2022).

[112] P. Runeson, M. Host, A. Rainer, and B. Regnell, *Case Study Research in Software Engineering: Guidelines and Examples*, 1st. Wiley Publishing, 2012, ISBN: 978-1-118-10435-4.

[113] A. Rahman, A. Partho, P. Morrison, and L. Williams, "What questions do programmers ask about configuration as code?" In *Proceedings of the 4th International Workshop on Rapid Continuous Software Engineering*, ser. RCoSE '18, Gothenburg, Sweden: Association for Computing Machinery, May 2018, pp. 16–22, ISBN: 978-1-4503-5745-6. DOI: 10.1145/3194760.3194769. [Online]. Available: https://doi.org/10.1145/3194760.3194769 (visited on 03/05/2020).

[114] T. Fritz and G. C. Murphy, "Using information fragments to answer the questions developers ask," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10, Cape Town, South Africa: Association for Computing Machinery, May 2010, pp. 175–184, ISBN: 978-1-60558-719-6. DOI: 10.1145/1806799.1806828. [Online]. Available: https://doi.org/10.1145/1806799.1806828 (visited on 03/05/2020).

[115] A. Ahmad, O. Leifler, and K. Sandahl, "The 36th ACM/SIGAPP Symposium on Applied Computing, March 22–26, 2021}{Virtual Event}," Republic of Korea, Mar. 2021, ISBN: 978-1-4503-8104-8. DOI: 10.1145/3412841.3442026.

[116] K. Y. Sharif and J. Buckley, "Observation of Open Source programmers' information seeking," in *2009 IEEE 17th International Conference on Program Comprehension*, ISSN: 1092-8138, May 2009, pp. 307–308. DOI: 10.1109/ICPC.2009.5090071.

[117] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik, "Communicative Intention in Code Review Questions," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, ISSN: 2576-3148, Sep. 2018, pp. 519–523. DOI: 10.1109/ICSME.2018.00061.

[118] T. D. LaToza and B. A. Myers, "Hard-to-answer questions about code," in *Evaluation and Usability of Programming Languages and Tools*, ser. PLATEAU '10, Reno, Nevada: Association for Computing Machinery, Oct. 2010, pp. 1–6, ISBN: 978-1-4503-0547-1. DOI: 10.1145/1937117.1937125. [Online]. Available: http://doi.org/10.1145/1937117.1937125 (visited on 04/06/2020).

[119] V. S. Sharma, R. Mehra, and V. Kaulgud, "What Do Developers Want? An Advisor Approach for Developer Priorities," in *2017 IEEE/ACM 10th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, May 2017, pp. 78–81. DOI: 10.1109/CHASE.2017.14.

[120] Q. Luo, F. Hariri, L. Eloussi, and D. Marinov, "An empirical analysis of flaky tests," en, ACM Press, 2014, pp. 643–653, ISBN: 978-1-4503-3056-5. DOI: 10.1145/2635868.2635920. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2635868.2635920 (visited on 05/12/2018).

[121] A. L. McNab and D. A. Ladd, "Information Quality: The Importance of Context and Trade-Offs," in *2014 47th Hawaii International Conference on System Sciences*, ISSN: 1530-1605, Jan. 2014, pp. 3525–3532. DOI: 10.1109/HICSS.2014.439.

[122] K. Ven, J. Verelst, and H. Mannaert, "Should You Adopt Open Source Software?" *IEEE Software*, vol. 25, no. 3, pp. 54–59, May 2008, Conference Name: IEEE Software, ISSN: 1937-4194. DOI: 10.1109/MS.2008.73.

[123] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: A survey," *Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67–120, 2012, ISSN: 1099-1689. DOI: 10.1002/stvr.430. [Online]. Available: http://dx.doi.org/10.1002/stvr.430.

[124] T. B. Noor and H. Hemmati, "A similarity-based approach for test case prioritization using historical failure data," in *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, Nov. 2015, pp. 58–68. DOI: 10.1109/ISSRE.2015.7381799.

[125] F. G. de Oliveira Neto, R. Torkar, and P. D. Machado, "Full modification coverage through automatic similarity-based test case selection," *Information and Software Technology*, vol. 80, pp. 124–137, 2016, ISSN: 0950-5849. DOI: http://dx.doi.org/10.1016/j.infsof.2016.08.008.

[126] H. Hemmati, A. Arcuri, and L. Briand, "Achieving scalable model-based testing through test case diversity," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 1, 6:1–6:42, Mar. 2013, ISSN: 1049-331X. DOI: 10.1145/2430536.2430540. [Online]. Available: http://doi.acm.org/10.1145/2430536.2430540.

[127] E. G. Cartaxo, P. D. L. Machado, and F. G. de Oliveira Neto, "On the use of a similarity function for test case selection in the context of model-based testing," *Software Testing, Verification and Reliability*, vol. 21, no. 2, pp. 75–100, 2011, ISSN: 1099-1689. DOI: 10.1002/stvr.413. [Online]. Available: http://dx.doi.org/10.1002/stvr.413.

[128] R. Feldt, S. Poulding, D. Clark, and S. Yoo, "Test set diameter: Quantifying the diversity of sets of test cases," in *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, Apr. 2016, pp. 223–233. DOI: 10.1109/ICST.2016.33.

[129] H. Hemmati, Z. Fang, and M. V. Mantyla, "Prioritizing manual test cases in traditional and rapid release environments," in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, Apr. 2015, pp. 1–10. DOI: 10.1109/ICST.2015.7102602.

[130] Y. Ledru, A. Petrenko, S. Boroday, and N. Mandran, "Prioritizing test cases with string distances," *Automated Software Engineering*, vol. 19, no. 1, pp. 65–95, Mar. 2012, ISSN: 1573-7535. DOI: 10.1007/s10515-011-0093-0. [Online]. Available: https://doi.org/10.1007/s10515-011-0093-0.

[131] C. Henard, M. Papadakis, M. Harman, Y. Jia, and Y. Le Traon, "Comparing white-box and black-box test prioritization," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, May 2016, pp. 523–534. DOI: 10.1145/2884781.2884791.

[132] R. Feldt, R. Torkar, T. Gorschek, and W. Afzal, "Searching for cognitively diverse tests: Towards universal test diversity metrics," in *2008 IEEE International Conference on Software Testing Verification and Validation Workshop*, Apr. 2008, pp. 178–186. DOI: 10.1109/ICSTW.2008.36.

[133] J. Campos, A. Arcuri, G. Fraser, and R. Abreu, "Continuous test generation: Enhancing continuous integration with automated test generation," in *IEEE/ACM Int. Conference on Automated Software Engineering (ASE)*, Västerås, Sweden: ACM, 2014, pp. 55–66.

[134] A. E. V. B. Coutinho, E. G. Cartaxo, and P. D. d. L. Machado, "Analysis of distance functions for similarity-based test suite reduction in the context of model-based testing," *Software Quality Journal*, vol. 24, no. 2, pp. 407–445, Jun. 2016, ISSN: 1573-1367. DOI: 10.1007/s11219-014-9265-z. [Online]. Available: https://doi.org/10.1007/s11219-014-9265-z.

[135] E. G. Cartaxo, F. G. de Oliveira Neto, and P. D. Machado, "Automated test case selection based on a similarity function.," *GI Jahrestagung (2)*, vol. 7, pp. 399–404, 2007.

[136] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 929–948, Oct. 2001, ISSN: 0098-5589. DOI: 10.1109/32.962562.

[137] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2008, ISSN: 1573-7616. DOI: 10.1007/s10664-008-9102-8.

[138] A. Marzal and E. Vidal, "Computation of normalized edit distance and applications," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 9, pp. 926–932, Sep. 1993, ISSN: 0162-8828. DOI: 10.1109/34.232078.

[139] F. G. de Oliveira Neto, R. Feldt, R. Torkar, and P. D. L. Machado, "Searching for models to evaluate software technology," in *Proceedings of the 1st International Workshop on Combining Modelling and Search-Based Software Engineering*, ser. CMSBSE '13, San Francisco, California: IEEE Press, 2013, pp. 12–15, ISBN: 978-1-4673-6284-9. [Online]. Available: http://dl.acm.org/citation.cfm?id=2662572.2662578.

[140] T. Gorschek, P. Garre, S. Larsson, and C. Wohlin, "A model for technology transfer in practice," *IEEE Software*, vol. 23, no. 6, pp. 88–95, Nov. 2006, ISSN: 0740-7459. DOI: 10.1109/MS.2006.147.

[141] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012, ISBN: 3642290434, 9783642290435.

[142] J. H. Andrews, L. C. Briand, and Y. Labiche, "Is mutation an appropriate tool for testing experiments?" In *Proceedings of the 27th International Conference on Software Engineering*, ser. ICSE '05, St. Louis, MO, USA: ACM, 2005, pp. 402–411, ISBN: 1-58113-963-2. DOI: 10.1145/1062455.1062530. [Online]. Available: http://doi.acm.org/10.1145/1062455.1062530.

[143] P. Duvall, S. Matyas, and A. Glover, *Continuous Integration: Improving Software Quality and Reducing Risk* (A Martin Fowler signature book). Addison-Wesley, 2007, ISBN: 9780321336385. [Online]. Available: https://books.google.se/books?id=MA8QmAEACAAJ.

[144] T. Mårtensson, D. Ståhl, and J. Bosch, "Test activities in the continuous integration and delivery pipeline," *Journal of Software: Evolution and Process*, vol. 31, no. 4, e2153, 2019.

[145] A. Ahmad, F. G. de Oliveira Neto, E. P. Enoiu, K. Sandahl, and O. Leifler, *Replication package for "An Industrial Study on the Challenges and Effects of Diversity-based Testing in Continuous Integration"*, version Version 1.0, Dec. 2020. DOI: 10.5281/zenodo.4305982. [Online]. Available: https://doi.org/10.5281/zenodo.4305982.

[146] O. S. Gómez, N. Juristo, and S. Vegas, "Replication, reproduction and re-analysis: Three ways for verifying experimental findings," in *Proceedings of the 1st international workshop on replication in empirical software engineering research (RESER 2010), Cape Town, South Africa*, 2010, pp. 1–7.

[147] R. Feldt, R. Torkar, T. Gorschek, and W. Afzal, "Searching for cognitively diverse tests: Towards universal test diversity metrics," in *2008 IEEE International Conference on Software Testing Verification and Validation Workshop*, Apr. 2008, pp. 178–186. DOI: 10.1109/ICSTW.2008.36.

[148] E. Knauss, M. Staron, W. Meding, O. Söder, A. Nilsson, and M. Castell, "Supporting Continuous Integration by Code-Churn Based Test Selection," in *2015 IEEE/ACM 2nd International Workshop on Rapid Continuous Software Engineering*, May 2015, pp. 19–25. DOI: 10.1109/RCoSE.2015.11.

[149] D. Marijan, M. Liaaen, and S. Sen, "DevOps Improvements for Reduced Cycle Times with Integrated Test Optimizations for Continuous Integration," in *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, Jul. 2018, pp. 22–27. DOI: 10.1109/COMPSAC.2018.00012.

[150] J. Liang, S. Elbaum, and G. Rothermel, "Redefining Prioritization: Continuous Prioritization for Continuous Integration," in *Proceedings of the 40th International Conference on Software Engineering*, ser. ICSE '18, New York, NY, USA: ACM, 2018, pp. 688–698, ISBN: 978-1-4503-5638-1. DOI: 10.1145/3180155.3180213. [Online]. Available: http://doi.acm.org/10.1145/3180155.3180213.

[151] J. H. Kwon and I. Y. Ko, "Cost-Effective Regression Testing Using Bloom Filters in Continuous Integration Development Environments," in *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, Dec. 2017, pp. 160–168. DOI: 10.1109/APSEC.2017.22.

[152] J.-M. Kim and A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments," in *Proceedings of the 24th International Conference on Software Engineering*, ser. ICSE '02, Orlando, Florida: ACM, 2002, pp. 119–129, ISBN: 1-58113-472-X. DOI: 10.1145/581339.581357. [Online]. Available: http://doi.acm.org/10.1145/581339.581357.

[153] Y. Zhu, E. Shihab, and P. C. Rigby, "Test re-prioritization in continuous testing environments," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Sep. 2018, pp. 69–79. DOI: 10.1109/ICSME.2018.00016.

[154] K. Petersen and E. Engström, "Finding relevant research solutions for practical problems: The serp taxonomy architecture," in *Proceedings of the 2014 International Workshop on Long-Term Industrial Collaboration on Software Engineering*, ser. WISE '14, Vasteras, Sweden: Association for Computing Machinery, 2014, pp. 13–20, ISBN: 9781450330459. DOI: 10.1145/2647648.2647650. [Online]. Available: https://doi.org/10.1145/2647648.2647650.

[155] N. Alshahwan and M. Harman, "Augmenting test suites effectiveness by increasing output diversity," in *2012 34th International Conference on Software Engineering (ICSE)*, 2012, pp. 1345–1348.

[156] P. M. Bueno, W. E. Wong, and M. Jino, "Improving random test sets using the diversity oriented test data generation," in *Proceedings of the 2nd international workshop on Random testing: co-located with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007)*, 2007, pp. 10–17.

[157] H. Hemmati, Z. Fang, M. V. Mäntylä, and B. Adams, "Prioritizing manual test cases in rapid release environments," *Software Testing, Verification and Reliability*, vol. 27, no. 6, pp. 1–10, 2017.

[158] E. Engström and K. Petersen, "Mapping software testing practice with software testing research — serp-test taxonomy," in *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2015, pp. 1–4. DOI: 10.1109/ICSTW.2015.7107470. [Online]. Available: https://doi.org/10.1109/ICSTW.2015.7107470.

[159] Y. Ledru, A. Petrenko, S. Boroday, and N. Mandran, "Prioritizing test cases with string distances," *Automated Software Engineering*, vol. 19, no. 1, pp. 65–95, 2012, ISSN: 1573-7535. DOI: 10.1007/s10515-011-0093-0.

[160] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Test case prioritization: An empirical study," in *Proceedings IEEE International Conference on Software Maintenance - 1999 (ICSM'99). 'Software Maintenance for Business Change' (Cat. No.99CB36360)*, 1999, pp. 179–188. DOI: 10.1109/ICSM.1999.792604. [Online]. Available: https://doi.org/10.1109/ICSM.1999.792604.

[161] H. Wickham, M. Averick, J. Bryan, W. Chang, L. D. McGowan, R. François, G. Grolemund, A. Hayes, L. Henry, J. Hester, M. Kuhn, T. L. Pedersen, E. Miller, S. M. Bache, K. Müller, J. Ooms, D. Robinson, D. P. Seidel, V. Spinu, K. Takahashi, D. Vaughan, C. Wilke, K. Woo, and H. Yutani, "Welcome to the tidyverse," *Journal of Open Source Software*, vol. 4, no. 43, p. 1686, 2019. DOI: 10.21105/joss.01686.

[162] S. M. Edwards, *Lemon: Freshing up your 'ggplot2' plots*, R package version 0.4.4, 2020. [Online]. Available: https://CRAN.R-project.org/package=lemon.

[163] M. Torchiano, *Effsize: Efficient effect size computation*, R package version 0.8.0, 2020. DOI: 10.5281/zenodo.1480624. [Online]. Available: https://CRAN.R-project.org/package=effsize.

[164] J. Gross and U. Ligges, *Nortest: Tests for normality*, R package version 1.0-4, 2015. [Online]. Available: https://CRAN.R-project.org/package=nortest.

[165] D. H. Ogle, P. Wheeler, and A. Dinno, *Fsa: Fisheries stock analysis*, R package version 0.8.30, 2020. [Online]. Available: https://github.com/droglenc/FSA.

[166] B. W. Yap and C. H. Sim, "Comparisons of various types of normality tests," *Journal of Statistical Computation and Simulation*, vol. 81, no. 12, pp. 2141–2155, 2011. DOI: 10.1080/00949655.2010.520163. [Online]. Available: https://doi.org/10.1080/00949655.2010.520163.

[167] G. Neumann, M. Harman, and S. Poulding, "Transformed vargha-delaney effect size," in *International Symposium on Search Based Software Engineering*, Springer, 2015, pp. 318–324.

[168] F. G. de Oliveira Neto, R. Torkar, and P. D. Machado, "Full modification coverage through automatic similarity-based test case selection," *Information and Software Technology*, vol. 80, pp. 124–137, 2016, ISSN: 0950-5849. DOI: http://dx.doi.org/10.1016/j.infsof.2016.08.008.

[169] A. E. V. B. Coutinho, E. G. Cartaxo, and P. D. L. Machado, "Analysis of distance functions for similarity-based test suite reduction in the context of model-based testing," *Software Quality Journal*, vol. 24, pp. , 407–445, Dec. 2016, ISSN: 0963-9314. DOI: 10.1007/s11219-014-9265-z. [Online]. Available: https://doi.org/10.1007/s11219-014-9265-z.

[170] T. C. Lethbridge, S. E. Sim, and J. Singer, "Studying software engineers: Data collection techniques for software field studies," *Empirical Software Engineering*, vol. 10, no. 3, pp. 311–341, Jul. 2005, ISSN: 1573-7616. DOI: 10.1007/s10664-005-1290-x. [Online]. Available: https://doi.org/10.1007/s10664-005-1290-x.

[171] J. Kontio, L. Lehtola, and J. Bragge, "Using the focus group method in software engineering: Obtaining practitioner and user experiences," in *Empirical Software Engineering, 2004. ISESE'04. Proceedings. 2004 International Symposium on*, IEEE, 2004, pp. 271–280.

[172] F. G. de Oliveira Neto, J. Horkoff, E. Knauss, R. Kasauli, and G. Liebel, "Challenges of aligning requirements engineering and system testing in large-scale agile: A multiple case study," in *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, 2017, pp. 315–322.

[173] E. Bjarnason, P. Runeson, M. Borg, M. Unterkalmsteiner, E. Engström, B. Regnell, G. Sabaliauskaite, A. Loconsole, T. Gorschek, and R. Feldt, "Challenges and practices in aligning requirements with verification and validation: A case study of six companies," *Empirical Software Engineering*, vol. 19, no. 6, pp. 1809–1855, Dec. 2014, ISSN: 1382-3256. DOI: 10.1007/s10664-013-9263-y.

[174] Y. S. Dai, M. Xie, K. L. Poh, and B. Yang, "Optimal testing-resource allocation with genetic algorithm for modular software systems," *Journal of Systems and Software*, vol. 66, no. 1, pp. 47–55, 2003, ISSN: 0164-1212. DOI: https://doi.org/10.1016/S0164-1212(02)00062-6. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121202000626.

[175] Z. Wang, K. Tang, and X. Yao, "Multi-objective approaches to optimal testing resource allocation in modular software systems," *IEEE Transactions on Reliability*, vol. 59, no. 3, pp. 563–575, 2010.

[176] S. Artzi, S. Kim, and M. D. Ernst, "Recrash: Making software failures reproducible by preserving object states," in *ECOOP 2008 – Object-Oriented Programming*, J. Vitek, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 542–565, ISBN: 978-3-540-70592-5.

[177] W. Jin and A. Orso, "Bugredux: Reproducing field failures for in-house debugging," in *2012 34th International Conference on Software Engineering (ICSE)*, 2012, pp. 474–484.

[178] J. Petke, M. B. Cohen, M. Harman, and S. Yoo, "Practical combinatorial interaction testing: Empirical findings on efficiency and early fault detection," *IEEE Transactions on Software Engineering*, vol. 41, no. 9, pp. 901–924, 2015.

[179] B. W. Boehm and P. N. Papaccio, "Understanding and controlling software costs," *IEEE transactions on software engineering*, vol. 14, no. 10, pp. 1462–1477, 1988.

[180] A. J. Ko, T. D. Latoza, and M. M. Burnett, "A practical guide to controlled experiments of software engineering tools with human participants," *Empirical Software Engineering*, vol. 20, no. 1, pp. 110–141, 2015.

[181] C. A. Furia, R. Feldt, and R. Torkar, "Bayesian data analysis in empirical software engineering research," *IEEE Transactions on Software Engineering*, 2019.

[182] A. Labuschagne, L. Inozemtseva, and R. Holmes, "Measuring the Cost of Regression Testing in Practice: A Study of Java Projects Using Continuous Integration," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017, event-place: Paderborn, Germany, New York, NY, USA: ACM, 2017, pp. 821–830, ISBN: 978-1-4503-5105-8. DOI: 10.1145/3106237.3106288. [Online]. Available: http://doi.acm.org/10.1145/3106237.3106288 (visited on 04/15/2019).

[183] M. Hilton, N. Nelson, T. Tunnell, D. Marinov, and D. Dig, "Trade-offs in Continuous Integration: Assurance, Security, and Flexibility," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017, event-place: Paderborn, Germany, New York, NY, USA: ACM, 2017, pp. 197–207, ISBN: 978-1-4503-5105-8. DOI: 10.1145/3106237.3106270. [Online]. Available: http://doi.acm.org/10.1145/3106237.3106270 (visited on 04/15/2019).

[184] J. Morán, C. Augusto, A. Bertolino, C. de la Riva, and J. Tuya, "Debugging Flaky Tests on Web Applications:" en, in *Proceedings of the 15th International Conference on Web Information Systems and Technologies*, Vienna, Austria: SCITEPRESS - Science and Technology Publications, 2019, pp. 454–461, ISBN: 978-989-758-386-5. DOI: 10.5220/0008559004540461. [Online]. Available: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0008559004540461 (visited on 09/08/2020).

[185] J. Morán, C. Augusto, A. Bertolino, C. D. L. Riva, and J. Tuya, "Flaky-Loc: Flakiness Localization for Reliable Test Suites in Web Applications," en, *Journal of Web Engineering*, pp. 267–296.–267–296. Jun. 2020, ISSN: 1544-5976. DOI: 10.13052/jwe1540-9589.1927. [Online]. Available: https://journals.riverpublishers.com/index.php/JWE/ (visited on 09/08/2020).

[186] Z. Dong, A. Tiwari, X. L. Yu, and A. Roychoudhury, "Concurrency-related Flaky Test Detection in Android apps," *arXiv:2005.10762 [cs]*, May 2020, arXiv: 2005.10762. [Online]. Available: http://arxiv.org/abs/2005.10762 (visited on 09/08/2020).

[187] M. A. Mascheroni and E. Irrazábal, "Identifying Key Success Factors in Stopping Flaky Tests in Automated REST Service Testing," en, *Journal of Computer Science and Technology*, vol. 18, no. 02, e16–e16, Oct. 2018, Number: 02, ISSN: 1666-6038. DOI: 10.24215/16666038.18.e16. [Online]. Available: https://journal.info.unlp.edu.ar/JCST/article/view/1087 (visited on 09/09/2020).

[188] A. Gambi, J. Bell, and A. Zeller, "Practical Test Dependency Detection," in *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, Apr. 2018, pp. 1–11. DOI: 10.1109/ICST.2018.00011.

[189] S. Dutta, A. Shi, R. Choudhary, Z. Zhang, A. Jain, and S. Misailovic, "Detecting flaky tests in probabilistic and machine learning applications," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2020, New York, NY, USA: Association for Computing Machinery, Jul. 2020, pp. 211–224, ISBN: 978-1-4503-8008-9. DOI: 10.1145/3395363.3397366. [On-

line]. Available: https://doi.org/10.1145/3395363.3397366 (visited on 09/08/2020).

[190] A. Shi, J. Bell, and D. Marinov, "Mitigating the effects of flaky tests on mutation testing," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2019, New York, NY, USA: Association for Computing Machinery, Jul. 2019, pp. 112–122, ISBN: 978-1-4503-6224-5. DOI: 10.1145/3293882.3330568. [Online]. Available: https://doi.org/10.1145/3293882.3330568 (visited on 09/08/2020).

[191] P. J. Fortier and H. Michel, *Computer Systems Performance Evaluation and Prediction*. USA: Butterworth-Heinemann, 2002, ISBN: 978-1-55558-260-9.

[192] D. Bowes, T. Hall, J. Petric, T. Shippey, and B. Turhan, "How Good Are My Tests?" In *2017 IEEE/ACM 8th Workshop on Emerging Trends in Software Metrics (WETSoM)*, May 2017, pp. 9–14. DOI: 10.1109/WETSoM.2017.2.

[193] A. Deursen, L. M. Moonen, A. Bergh, and G. Kok, "Refactoring Test Code," CWI (Centre for Mathematics and Computer Science), Amsterdam, The Netherlands, Tech. Rep., 2001.

[194] C. Kaner, "What Is a Good Test Case?" en, *Software Testing Analysis & Review Conference (STAR) East, Orlando, FL, May 12-16, 2003*, p. 16,

[195] A. Beer, M. Junker, H. Femmer, and M. Felderer, "Initial Investigations on the Influence of Requirement Smells on Test-Case Design," in *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, Sep. 2017, pp. 323–326. DOI: 10.1109/REW.2017.43.

[196] *Factor definition and meaning | Collins English Dictionary*, en. [Online]. Available: https://www.collinsdictionary.com/dictionary/english/factor (visited on 02/12/2021).

[197] V. Garousi and B. Küçük, "Smells in software test code: A survey of knowledge in industry and academia," en, *Journal of Systems and Software*, vol. 138, pp. 52–81, Apr. 2018, ISSN: 0164-1212. DOI: 10.1016/j.jss.2017.12.013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121217303060 (visited on 01/22/2020).

[198] H. S. Dar, "Reducing Ambiguity in Requirements Elicitation via Gamification," in *2020 IEEE 28th International Requirements Engineering Conference (RE)*, ISSN: 2332-6441, Aug. 2020, pp. 440–444. DOI: 10.1109/RE48521.2020.00065.

[199] C. Ziftci and D. Cavalcanti, "De-Flake Your Tests : Automatically Locating Root Causes of Flaky Tests in Code At Google," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, ISSN: 2576-3148, Sep. 2020, pp. 736–745. DOI: 10.1109/ICSME46990.2020.00083.

[200] K. Presler-Marshall, E. Horton, S. Heckman, and K. Stolee, "Wait, Wait. No, Tell Me. Analyzing Selenium Configuration Effects on Test Flakiness," in *2019 IEEE/ACM 14th International Workshop on Automation of Software Test (AST)*, May 2019, pp. 7–13. DOI: 10.1109/AST.2019.000-1.

[201] W. Lam, S. Winter, A. Astorga, V. Stodden, and D. Marinov, "Understanding Reproducibility and Characteristics of Flaky Tests Through Test Reruns in Java Projects," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, ISSN: 2332-6549, Oct. 2020, pp. 403–413. DOI: 10.1109/ISSRE5003.2020.00045.

[202] A. Groce and J. Holmes, "Practical Automatic Lightweight Nondeterminism and Flaky Test Detection and Debugging for Python," in *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*, Dec. 2020, pp. 188–195. DOI: 10.1109/QRS51102.2020.00035.

[203] E. Kowalczyk, K. Nair, Z. Gao, L. Silberstein, T. Long, and A. Memon, "Modeling and Ranking Flaky Tests at Apple," in *2020 IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, Oct. 2020, pp. 110–119.

[204] N. S. Altman, "An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992, Publisher: [American Statistical Association, Taylor & Francis, Ltd.], ISSN: 0003-1305. DOI: 10.2307/2685209. [Online]. Available: https://www.jstor.org/stable/2685209 (visited on 12/29/2020).

[205] F. Palomba and A. Zaidman, "Does Refactoring of Test Smells Induce Fixing Flaky Tests?" In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Sep. 2017, pp. 1–12. DOI: 10.1109/ICSME.2017.12.

[206] R. Shams and R. E. Mercer, "Classifying Spam Emails Using Text and Readability Features," in *2013 IEEE 13th International Conference on Data Mining*, ISSN: 2374-8486, Dec. 2013, pp. 657–666. DOI: 10.1109/ICDM.2013.131.

[207] S. K. Tuteja and N. Bogiri, "Email Spam filtering using BPNN classification algorithm," in *2016 International Conference on Automatic Control and Dynamic Optimization Techniques (ICACDOT)*, ISSN: null, Sep. 2016, pp. 915–919. DOI: 10.1109/ICACDOT.2016.7877720.

[208] E. Sahın, M. Aydos, and F. Orhan, "Spam/ham e-mail classification using machine learning methods based on bag of words technique," in *2018 26th Signal Processing and Communications Applications Conference (SIU)*, ISSN: null, May 2018, pp. 1–4. DOI: 10.1109/SIU.2018.8404347.

[209] K. Mathew and B. Issac, "Intelligent spam classification for mobile text message," in *Proceedings of 2011 International Conference on Computer Science and Network Technology*, ISSN: null, vol. 1, Dec. 2011, pp. 101–105. DOI: 10.1109/ICCSNT.2011.6181918.

[210] A. B. M. S. Ali and Y. Xiang, "Spam Classification Using Adaptive Boosting Algorithm," in *6th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007)*, ISSN: null, Jul. 2007, pp. 972–976. DOI: 10.1109/ICIS.2007.170.

[211] A. A. Alurkar, S. B. Ranade, S. V. Joshi, S. S. Ranade, P. A. Sonewar, P. N. Mahalle, and A. V. Deshpande, "A proposed data science approach for email spam classification using machine learning techniques," in *2017 Internet of Things Business Models, Users, and Networks*, ISSN: null, Nov. 2017, pp. 1–5. DOI: 10.1109/CTTE.2017.8260935.

[212] S. Vahora, M. Hasan, and R. Lakhani, "Novel approach: Naïve Bayes with Vector space model for spam classification," in *2011 Nirma University International Conference on Engineering*, ISSN: 2375-1282, Dec. 2011, pp. 1–5. DOI: 10.1109/NUiConE.2011.6153245.

[213] T.-Y. Yu and W.-C. Hsu, "E-mail Spam Filtering Using Support Vector Machines with Selection of Kernel Function Parameters," in *2009 Fourth International Conference on Innovative Computing, Information and Control (ICICIC)*, ISSN: null, Dec. 2009, pp. 764–767. DOI: 10.1109/ICICIC.2009.184.

[214] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *Proceedings of the 23rd international conference on Machine learning*, ser. ICML '06, Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, Jun. 2006, pp. 161–168, ISBN: 978-1-59593-383-6. DOI: 10.1145/1143844.1143865. [Online]. Available: https://doi.org/10.1145/1143844.1143865 (visited on 01/21/2020).

[215] C.-Y. Chiu and Y.-T. Huang, "Integration of Support Vector Machine with Naïve Bayesian Classifier for Spam Classification," in *Fourth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2007)*, ISSN: null, vol. 1, Aug. 2007, pp. 618–622. DOI: 10.1109/FSKD.2007.366.

[216] Z. Jia, W. Li, W. Gao, and Y. Xia, "Research on Web Spam Detection Based on Support Vector Machine," in *2012 International Conference on Communication Systems and Network Technologies*, ISSN: null, May 2012, pp. 517–520. DOI: 10.1109/CSNT.2012.117.

[217] A. S. Katasev, L. Y. Emaletdinova, and D. V. Kataseva, "Neural Network Spam Filtering Technology," in *2018 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM)*, ISSN: null, May 2018, pp. 1–5. DOI: 10.1109/ICIEAM.2018.8728862.

[218] M. K. and R. Kumar, "Spam Mail Classification Using Combined Approach of Bayesian and Neural Network," in *2010 International Conference on Computational Intelligence and Communication Networks*, ISSN: null, Nov. 2010, pp. 145–149. DOI: 10.1109/CICN.2010.39.

[219] L. Firte, C. Lemnaru, and R. Potolea, "Spam detection filter using KNN algorithm and resampling," in *Proceedings of the 2010 IEEE 6th International Conference on Intelligent Computer Communication and Processing*, ISSN: null, Aug. 2010, pp. 27–33. DOI: 10.1109/ICCP.2010.5606466.

[220] *Weka 3 - Data Mining with Open Source Machine Learning Software in Java*. [Online]. Available: https://www.cs.waikato.ac.nz/ml/weka/index.html (visited on 10/12/2020).

Papers

The papers associated with this thesis have been removed for copyright reasons. For more details about these see:

https://doi.org/10.3384/9789179294236

Department of Computer and Information Science
Linköpings universitet

Dissertations

Linköping Studies in Science and Technology
Linköping Studies in Arts and Sciences
Linköping Studies in Statistics
Linköping Studies in Information Science

Linköping Studies in Science and Technology

No 14 **Anders Haraldsson:** A Program Manipulation System Based on Partial Evaluation, 1977, ISBN 91-7372-144-1.

No 17 **Bengt Magnhagen:** Probability Based Verification of Time Margins in Digital Designs, 1977, ISBN 91-7372-157-3.

No 18 **Mats Cedwall**: Semantisk analys av process-beskrivningar i naturligt språk, 1977, ISBN 91- 7372-168-9.

No 22 **Jaak Urmi:** A Machine Independent LISP Compiler and its Implications for Ideal Hardware, 1978, ISBN 91-7372-188-3.

No 33 **Tore Risch:** Compilation of Multiple File Queries in a Meta-Database System, 1978, ISBN 91- 7372-232-4.

No 51 **Erland Jungert:** Synthesizing Database Structures from a User Oriented Data Model, 1980, ISBN 91-7372-387-8.

No 54 **Sture Hägglund:** Contributions to the Development of Methods and Tools for Interactive Design of Applications Software, 1980, ISBN 91-7372-404-1.

No 55 **Pär Emanuelson:** Performance Enhancement in a Well-Structured Pattern Matcher through Partial Evaluation, 1980, ISBN 91-7372-403-3.

No 58 **Bengt Johnsson, Bertil Andersson:** The Human-Computer Interface in Commercial Systems, 1981, ISBN 91-7372-414-9.

No 69 **H. Jan Komorowski:** A Specification of an Abstract Prolog Machine and its Application to Partial Evaluation, 1981, ISBN 91-7372-479-3.

No 71 **René Reboh:** Knowledge Engineering Techniques and Tools for Expert Systems, 1981, ISBN 91-7372-489-0.

No 77 **Östen Oskarsson:** Mechanisms of Modifiability in large Software Systems, 1982, ISBN 91- 7372-527-7.

No 94 **Hans Lunell:** Code Generator Writing Systems, 1983, ISBN 91-7372-652-4.

No 97 **Andrzej Lingas:** Advances in Minimum Weight Triangulation, 1983, ISBN 91-7372-660-5.

No 109 **Peter Fritzson:** Towards a Distributed Programming Environment based on Incremental Compilation, 1984, ISBN 91-7372-801-2.

No 111 **Erik Tengvald:** The Design of Expert Planning Systems. An Experimental Operations Planning System for Turning, 1984, ISBN 91-7372- 805-5.

No 155 **Christos Levcopoulos:** Heuristics for Minimum Decompositions of Polygons, 1987, ISBN 91-7870-133-3.

No 165 **James W. Goodwin:** A Theory and System for Non-Monotonic Reasoning, 1987, ISBN 91-7870-183-X.

No 170 **Zebo Peng:** A Formal Methodology for Automated Synthesis of VLSI Systems, 1987, ISBN 91-7870-225-9.

No 174 **Johan Fagerström:** A Paradigm and System for Design of Distributed Systems, 1988, ISBN 91-7870-301-8.

No 192 **Dimiter Driankov:** Towards a Many Valued Logic of Quantified Belief, 1988, ISBN 91-7870-374-3.

No 213 **Lin Padgham:** Non-Monotonic Inheritance for an Object Oriented Knowledge Base, 1989, ISBN 91-7870-485-5.

No 214 **Tony Larsson:** A Formal Hardware Description and Verification Method, 1989, ISBN 91-7870-517-7.

No 221 **Michael Reinfrank:** Fundamentals and Logical Foundations of Truth Maintenance, 1989, ISBN 91-7870-546-0.

No 239 **Jonas Löwgren:** Knowledge-Based Design Support and Discourse Management in User Interface Management Systems, 1991, ISBN 91-7870-720-X.

No 244 **Henrik Eriksson:** Meta-Tool Support for Knowledge Acquisition, 1991, ISBN 91-7870-746-3.

No 252 **Peter Eklund:** An Epistemic Approach to Interactive Design in Multiple Inheritance Hierarchies, 1991, ISBN 91-7870-784-6.

No 258 **Patrick Doherty:** NML3 - A Non-Monotonic Formalism with Explicit Defaults, 1991, ISBN 91-7870-816-8.

No 260 **Nahid Shahmehri:** Generalized Algorithmic Debugging, 1991, ISBN 91-7870-828-1.

No 264 **Nils Dahlbäck:** Representation of Discourse-Cognitive and Computational Aspects, 1992, ISBN 91-7870-850-8.

No 265 **Ulf Nilsson:** Abstract Interpretations and Abstract Machines: Contributions to a Methodology for the Implementation of Logic Programs, 1992, ISBN 91-7870-858-3.

No 270 **Ralph Rönnquist:** Theory and Practice of Tense-bound Object References, 1992, ISBN 91-7870-873-7.

No 273 **Björn Fjellborg:** Pipeline Extraction for VLSI Data Path Synthesis, 1992, ISBN 91-7870-880-X.

No 276 **Staffan Bonnier:** A Formal Basis for Horn Clause Logic with External Polymorphic Functions, 1992, ISBN 91-7870-896-6.

No 277 **Kristian Sandahl:** Developing Knowledge Management Systems with an Active Expert Methodology, 1992, ISBN 91-7870-897-4.

No 281 **Christer Bäckström:** Computational Complexity of Reasoning about Plans, 1992, ISBN 91-7870-979-2.

No 292 **Mats Wirén:** Studies in Incremental Natural Language Analysis, 1992, ISBN 91-7871-027-8.

No 297 **Mariam Kamkar:** Interprocedural Dynamic Slicing with Applications to Debugging and Testing, 1993, ISBN 91-7871-065-0.

No 302 **Tingting Zhang:** A Study in Diagnosis Using Classification and Defaults, 1993, ISBN 91-7871-078-2.

No 312 **Arne Jönsson:** Dialogue Management for Natural Language Interfaces - An Empirical Approach, 1993, ISBN 91-7871-110-X.

No 338 **Simin Nadjm-Tehrani:** Reactive Systems in Physical Environments: Compositional Modelling and Framework for Verification, 1994, ISBN 91-7871-237-8.

No 371 **Bengt Savén:** Business Models for Decision Support and Learning. A Study of Discrete-Event Manufacturing Simulation at Asea/ABB 1968-1993, 1995, ISBN 91-7871-494-X.

No 375 **Ulf Söderman:** Conceptual Modelling of Mode Switching Physical Systems, 1995, ISBN 91-7871-516-4.

No 383 **Andreas Kågedal:** Exploiting Groundness in Logic Programs, 1995, ISBN 91-7871-538-5.

No 396 **George Fodor:** Ontological Control, Description, Identification and Recovery from Problematic Control Situations, 1995, ISBN 91-7871-603-9.

No 413 **Mikael Pettersson:** Compiling Natural Semantics, 1995, ISBN 91-7871-641-1.

No 414 **Xinli Gu:** RT Level Testability Improvement by Testability Analysis and Transformations, 1996, ISBN 91-7871-654-3.

No 416 **Hua Shu:** Distributed Default Reasoning, 1996, ISBN 91-7871-665-9.

No 429 **Jaime Villegas:** Simulation Supported Industrial Training from an Organisational Learning Perspective - Development and Evaluation of the SSIT Method, 1996, ISBN 91-7871-700-0.

No 431 **Peter Jonsson:** Studies in Action Planning: Algorithms and Complexity, 1996, ISBN 91-7871-704-3.

No 437 **Johan Boye:** Directional Types in Logic Programming, 1996, ISBN 91-7871-725-6.

No 439 **Cecilia Sjöberg:** Activities, Voices and Arenas: Participatory Design in Practice, 1996, ISBN 91-7871-728-0.

No 448 **Patrick Lambrix:** Part-Whole Reasoning in Description Logics, 1996, ISBN 91-7871-820-1.

No 452 **Kjell Orsborn:** On Extensible and Object-Relational Database Technology for Finite Element Analysis Applications, 1996, ISBN 91-7871-827-9.

No 459 **Olof Johansson:** Development Environments for Complex Product Models, 1996, ISBN 91-7871-855-4.

No 461 **Lena Strömbäck:** User-Defined Constructions in Unification-Based Formalisms, 1997, ISBN 91-7871-857-0.

No 462 **Lars Degerstedt:** Tabulation-based Logic Programming: A Multi-Level View of Query Answering, 1996, ISBN 91-7871-858-9.

No 475 **Fredrik Nilsson:** Strategi och ekonomisk styrning - En studie av hur ekonomiska styrsystem utformas och används efter företagsförvärv, 1997, ISBN 91-7871-914-3.

No 480 **Mikael Lindvall:** An Empirical Study of Requirements-Driven Impact Analysis in Object-Oriented Software Evolution, 1997, ISBN 91-7871-927-5.

No 485 **Göran Forslund:** Opinion-Based Systems: The Cooperative Perspective on Knowledge-Based Decision Support, 1997, ISBN 91-7871-938-0.

No 494 **Martin Sköld:** Active Database Management Systems for Monitoring and Control, 1997, ISBN 91-7219-002-7.

No 495 **Hans Olsén:** Automatic Verification of Petri Nets in a CLP framework, 1997, ISBN 91-7219-011-6.

No 498 **Thomas Drakengren:** Algorithms and Complexity for Temporal and Spatial Formalisms, 1997, ISBN 91-7219-019-1.

No 502 **Jakob Axelsson:** Analysis and Synthesis of Heterogeneous Real-Time Systems, 1997, ISBN 91-7219-035-3.

No 503 **Johan Ringström:** Compiler Generation for Data-Parallel Programming Languages from Two-Level Semantics Specifications, 1997, ISBN 91-7219-045-0.

No 512 **Anna Moberg:** Närhet och distans - Studier av kommunikationsmönster i satellitkontor och flexibla kontor, 1997, ISBN 91-7219-119-8.

No 520 **Mikael Ronström:** Design and Modelling of a Parallel Data Server for Telecom Applications, 1998, ISBN 91-7219-169-4.

No 522 **Niclas Ohlsson:** Towards Effective Fault Prevention - An Empirical Study in Software Engineering, 1998, ISBN 91-7219-176-7.

No 526 **Joachim Karlsson:** A Systematic Approach for Prioritizing Software Requirements, 1998, ISBN 91-7219-184-8.

No 530 **Henrik Nilsson:** Declarative Debugging for Lazy Functional Languages, 1998, ISBN 91-7219-197-X.

No 555 **Jonas Hallberg:** Timing Issues in High-Level Synthesis, 1998, ISBN 91-7219-369-7.

No 561 **Ling Lin:** Management of 1-D Sequence Data - From Discrete to Continuous, 1999, ISBN 91-7219-402-2.

No 563 **Eva L Ragnemalm:** Student Modelling based on Collaborative Dialogue with a Learning Companion, 1999, ISBN 91-7219-412-X.

No 567 **Jörgen Lindström:** Does Distance matter? On geographical dispersion in organisations, 1999, ISBN 91-7219-439-1.

No 582 **Vanja Josifovski:** Design, Implementation and Evaluation of a Distributed Mediator System for Data Integration, 1999, ISBN 91-7219-482-0.

No 589 **Rita Kovordányi:** Modeling and Simulating Inhibitory Mechanisms in Mental Image Reinterpretation - Towards Cooperative Human-Computer Creativity, 1999, ISBN 91-7219-506-1.

No 592 **Mikael Ericsson:** Supporting the Use of Design Knowledge - An Assessment of Commenting Agents, 1999, ISBN 91-7219-532-0.

No 593 **Lars Karlsson:** Actions, Interactions and Narratives, 1999, ISBN 91-7219-534-7.

No 594 **C. G. Mikael Johansson:** Social and Organizational Aspects of Requirements Engineering Methods - A practice-oriented approach, 1999, ISBN 91-7219-541-X.

No 595 **Jörgen Hansson:** Value-Driven Multi-Class Overload Management in Real-Time Database Systems, 1999, ISBN 91-7219-542-8.

No 596 **Niklas Hallberg:** Incorporating User Values in the Design of Information Systems and Services in the Public Sector: A Methods Approach, 1999, ISBN 91-7219-543-6.

No 597 **Vivian Vimarlund:** An Economic Perspective on the Analysis of Impacts of Information Technology: From Case Studies in Health-Care towards General Models and Theories, 1999, ISBN 91-7219-544-4.

No 598 **Johan Jenvald:** Methods and Tools in Computer-Supported Taskforce Training, 1999, ISBN 91-7219-547-9.

No 607 **Magnus Merkel:** Understanding and enhancing translation by parallel text processing, 1999, ISBN 91-7219-614-9.

No 611 **Silvia Coradeschi:** Anchoring symbols to sensory data, 1999, ISBN 91-7219-623-8.

No 613 **Man Lin:** Analysis and Synthesis of Reactive Systems: A Generic Layered Architecture Perspective, 1999, ISBN 91-7219-630-0.

No 618 **Jimmy Tjäder:** Systemimplementering i praktiken - En studie av logiker i fyra projekt, 1999, ISBN 91-7219-657-2.

No 627 **Vadim Engelson:** Tools for Design, Interactive Simulation, and Visualization of Object-Oriented Models in Scientific Computing, 2000, ISBN 91-7219-709-9.

No 637 **Esa Falkenroth:** Database Technology for Control and Simulation, 2000, ISBN 91-7219-766-8.

No 639 **Per-Arne Persson:** Bringing Power and Knowledge Together: Information Systems Design for Autonomy and Control in Command Work, 2000, ISBN 91-7219-796-X.

No 660 **Erik Larsson:** An Integrated System-Level Design for Testability Methodology, 2000, ISBN 91-7219-890-7.

No 688 **Marcus Bjäreland:** Model-based Execution Monitoring, 2001, ISBN 91-7373-016-5.

No 689 **Joakim Gustafsson:** Extending Temporal Action Logic, 2001, ISBN 91-7373-017-3.

No 720 **Carl-Johan Petri:** Organizational Information Provision - Managing Mandatory and Discretionary Use of Information Technology, 2001, ISBN 91-7373-126-9.

No 724 **Paul Scerri:** Designing Agents for Systems with Adjustable Autonomy, 2001, ISBN 91-7373-207-9.

No 725 **Tim Heyer:** Semantic Inspection of Software Artifacts: From Theory to Practice, 2001, ISBN 91-7373-208-7.

No 726 **Pär Carlshamre:** A Usability Perspective on Requirements Engineering - From Methodology to Product Development, 2001, ISBN 91-7373-212-5.

No 732 **Juha Takkinen:** From Information Management to Task Management in Electronic Mail, 2002, ISBN 91-7373-258-3.

No 745 **Johan Åberg:** Live Help Systems: An Approach to Intelligent Help for Web Information Systems, 2002, ISBN 91-7373-311-3.

No 746 **Rego Granlund:** Monitoring Distributed Teamwork Training, 2002, ISBN 91-7373-312-1.

No 757 **Henrik André-Jönsson:** Indexing Strategies for Time Series Data, 2002, ISBN 917373-346-6.

No 747 **Anneli Hagdahl:** Development of IT-supported Interorganisational Collaboration - A Case Study in the Swedish Public Sector, 2002, ISBN 91-7373-314-8.

No 749 **Sofie Pilemalm:** Information Technology for Non-Profit Organisations - Extended Participatory Design of an Information System for Trade Union Shop Stewards, 2002, ISBN 91-7373-318-0.

No 765 **Stefan Holmlid:** Adapting users: Towards a theory of use quality, 2002, ISBN 91-7373-397-0.

No 771 **Magnus Morin:** Multimedia Representations of Distributed Tactical Operations, 2002, ISBN 91-7373-421-7.

No 772 **Pawel Pietrzak:** A Type-Based Framework for Locating Errors in Constraint Logic Programs, 2002, ISBN 91-7373-422-5.

No 758 **Erik Berglund:** Library Communication Among Programmers Worldwide, 2002, ISBN 91-7373-349-0.

No 774 **Choong-ho Yi:** Modelling Object-Oriented Dynamic Systems Using a Logic-Based Framework, 2002, ISBN 91-7373-424-1.

No 779 **Mathias Broxvall:** A Study in the Computational Complexity of Temporal Reasoning, 2002, ISBN 91-7373-440-3.

No 793 **Asmus Pandikow:** A Generic Principle for Enabling Interoperability of Structured and Object-Oriented Analysis and Design Tools, 2002, ISBN 91-7373-479-9.

No 785 **Lars Hult:** Publika Informationstjänster. En studie av den Internetbaserade encyklopedins bruksegenskaper, 2003, ISBN 91-7373-461-6.

No 800 **Lars Taxén:** A Framework for the Coordination of Complex Systems´ Development, 2003, ISBN 91-7373-604-X.

No 808 **Klas Gäre:** Tre perspektiv på förväntningar och förändringar i samband med införande av informationssystem, 2003, ISBN 91-7373-618-X.

No 821 **Mikael Kindborg:** Concurrent Comics - programming of social agents by children, 2003, ISBN 91-7373-651-1.

No 823 **Christina Ölvingson:** On Development of Information Systems with GIS Functionality in Public Health Informatics: A Requirements Engineering Approach, 2003, ISBN 91-7373-656-2.

No 828 **Tobias Ritzau:** Memory Efficient Hard Real-Time Garbage Collection, 2003, ISBN 91-7373-666-X.

No 833 **Paul Pop:** Analysis and Synthesis of Communication-Intensive Heterogeneous Real-Time Systems, 2003, ISBN 91-7373-683-X.

No 852 **Johan Moe:** Observing the Dynamic Behaviour of Large Distributed Systems to Improve Development and Testing – An Empirical Study in Software Engineering, 2003, ISBN 91-7373-779-8.

No 867 **Erik Herzog:** An Approach to Systems Engineering Tool Data Representation and Exchange, 2004, ISBN 91-7373-929-4.

No 872 **Aseel Berglund:** Augmenting the Remote Control: Studies in Complex Information Navigation for Digital TV, 2004, ISBN 91-7373-940-5.

No 869 **Jo Skåmedal:** Telecommuting's Implications on Travel and Travel Patterns, 2004, ISBN 91-7373-935-9.

No 870 **Linda Askenäs:** The Roles of IT - Studies of Organising when Implementing and Using Enterprise Systems, 2004, ISBN 91-7373-936-7.

No 874 **Annika Flycht-Eriksson:** Design and Use of Ontologies in Information-Providing Dialogue Systems, 2004, ISBN 91-7373-947-2.

No 873 **Peter Bunus:** Debugging Techniques for Equation-Based Languages, 2004, ISBN 91-7373-941-3.

No 876 **Jonas Mellin:** Resource-Predictable and Efficient Monitoring of Events, 2004, ISBN 91-7373-956-1.

No 883 **Magnus Bång:** Computing at the Speed of Paper: Ubiquitous Computing Environments for Healthcare Professionals, 2004, ISBN 91-7373-971-5.

No 882 **Robert Eklund:** Disfluency in Swedish human-human and human-machine travel booking dialogues, 2004, ISBN 91-7373-966-9.

No 887 **Anders Lindström:** English and other Foreign Linguistic Elements in Spoken Swedish. Studies of Productive Processes and their Modelling using Finite-State Tools, 2004, ISBN 91-7373-981-2.

No 889 **Zhiping Wang:** Capacity-Constrained Production-inventory systems - Modelling and Analysis in both a traditional and an e-business context, 2004, ISBN 91-85295-08-6.

No 893 **Pernilla Qvarfordt:** Eyes on Multimodal Interaction, 2004, ISBN 91-85295-30-2.

No 910 **Magnus Kald:** In the Borderland between Strategy and Management Control - Theoretical Framework and Empirical Evidence, 2004, ISBN 91-85295-82-5.

No 918 **Jonas Lundberg:** Shaping Electronic News: Genre Perspectives on Interaction Design, 2004, ISBN 91-85297-14-3.

No 900 **Mattias Arvola:** Shades of use: The dynamics of interaction design for sociable use, 2004, ISBN 91-85295-42-6.

No 920 **Luis Alejandro Cortés:** Verification and Scheduling Techniques for Real-Time Embedded Systems, 2004, ISBN 91-85297-21-6.

No 929 **Diana Szentivanyi:** Performance Studies of Fault-Tolerant Middleware, 2005, ISBN 91-85297-58-5.

No 933 **Mikael Cäker:** Management Accounting as Constructing and Opposing Customer Focus: Three Case Studies on Management Accounting and Customer Relations, 2005, ISBN 91-85297-64-X.

No 937 **Jonas Kvarnström:** TALplanner and Other Extensions to Temporal Action Logic, 2005, ISBN 91-85297-75-5.

No 938 **Bourhane Kadmiry:** Fuzzy Gain-Scheduled Visual Servoing for Unmanned Helicopter, 2005, ISBN 91-85297-76-3.

No 945 **Gert Jervan:** Hybrid Built-In Self-Test and Test Generation Techniques for Digital Systems, 2005, ISBN 91-85297-97-6.

No 946 **Anders Arpteg:** Intelligent Semi-Structured Information Extraction, 2005, ISBN 91-85297-98-4.

No 947 **Ola Angelsmark:** Constructing Algorithms for Constraint Satisfaction and Related Problems - Methods and Applications, 2005, ISBN 91-85297-99-2.

No 963 **Calin Curescu:** Utility-based Optimisation of Resource Allocation for Wireless Networks, 2005, ISBN 91-85457-07-8.

No 972 **Björn Johansson:** Joint Control in Dynamic Situations, 2005, ISBN 91-85457-31-0.

No 974 **Dan Lawesson:** An Approach to Diagnosability Analysis for Interacting Finite State Systems, 2005, ISBN 91-85457-39-6.

No 979 **Claudiu Duma:** Security and Trust Mechanisms for Groups in Distributed Services, 2005, ISBN 91-85457-54-X.

No 983 **Sorin Manolache:** Analysis and Optimisation of Real-Time Systems with Stochastic Behaviour, 2005, ISBN 91-85457-60-4.

No 986 **Yuxiao Zhao:** Standards-Based Application Integration for Business-to-Business Communications, 2005, ISBN 91-85457-66-3.

No 1004 **Patrik Haslum:** Admissible Heuristics for Automated Planning, 2006, ISBN 91-85497-28-2.

No 1005 **Aleksandra Tešanovic:** Developing Reusable and Reconfigurable Real-Time Software using Aspects and Components, 2006, ISBN 91-85497-29-0.

No 1008 **David Dinka:** Role, Identity and Work: Extending the design and development agenda, 2006, ISBN 91-85497-42-8.

No 1009 **Iakov Nakhimovski:** Contributions to the Modeling and Simulation of Mechanical Systems with Detailed Contact Analysis, 2006, ISBN 91-85497-43-X.

No 1013 **Wilhelm Dahllöf:** Exact Algorithms for Exact Satisfiability Problems, 2006, ISBN 91-85523-97-6.

No 1016 **Levon Saldamli:** PDEModelica - A High-Level Language for Modeling with Partial Differential Equations, 2006, ISBN 91-85523-84-4.

No 1017 **Daniel Karlsson:** Verification of Component-based Embedded System Designs, 2006, ISBN 91-85523-79-8

No 1018 **Ioan Chisalita:** Communication and Networking Techniques for Traffic Safety Systems, 2006, ISBN 91-85523-77-1.

No 1019 **Tarja Susi:** The Puzzle of Social Activity - The Significance of Tools in Cognition and Cooperation, 2006, ISBN 91-85523-71-2.

No 1021 **Andrzej Bednarski:** Integrated Optimal Code Generation for Digital Signal Processors, 2006, ISBN 91-85523-69-0.

No 1022 **Peter Aronsson:** Automatic Parallelization of Equation-Based Simulation Programs, 2006, ISBN 91-85523-68-2.

No 1030 **Robert Nilsson:** A Mutation-based Framework for Automated Testing of Timeliness, 2006, ISBN 91-85523-35-6.

No 1034 **Jon Edvardsson:** Techniques for Automatic Generation of Tests from Programs and Specifications, 2006, ISBN 91-85523-31-3.

No 1035 **Vaida Jakoniene:** Integration of Biological Data, 2006, ISBN 91-85523-28-3.

No 1045 **Genevieve Gorrell:** Generalized Hebbian Algorithms for Dimensionality Reduction in Natural Language Processing, 2006, ISBN 91-85643-88-2.

No 1051 **Yu-Hsing Huang:** Having a New Pair of Glasses - Applying Systemic Accident Models on Road Safety, 2006, ISBN 91-85643-64-5.

No 1054 **Åsa Hedenskog:** Perceive those things which cannot be seen - A Cognitive Systems Engineering perspective on requirements management, 2006, ISBN 91-85643-57-2.

No 1061 **Cécile Åberg:** An Evaluation Platform for Semantic Web Technology, 2007, ISBN 91-85643-31-9.

No 1073 **Mats Grindal:** Handling Combinatorial Explosion in Software Testing, 2007, ISBN 978-91-85715-74-9.

No 1075 **Almut Herzog:** Usable Security Policies for Runtime Environments, 2007, ISBN 978-91-85715-65-7.

No 1079 **Magnus Wahlström:** Algorithms, measures, and upper bounds for Satisfiability and related problems, 2007, ISBN 978-91-85715-55-8.

No 1083 **Jesper Andersson:** Dynamic Software Architectures, 2007, ISBN 978-91-85715-46-6.

No 1086 **Ulf Johansson:** Obtaining Accurate and Comprehensible Data Mining Models - An Evolutionary Approach, 2007, ISBN 978-91-85715-34-3.

No 1089 **Traian Pop:** Analysis and Optimisation of Distributed Embedded Systems with Heterogeneous Scheduling Policies, 2007, ISBN 978-91-85715-27-5.

No 1091 **Gustav Nordh:** Complexity Dichotomies for CSP-related Problems, 2007, ISBN 978-91-85715-20-6.

No 1106 **Per Ola Kristensson:** Discrete and Continuous Shape Writing for Text Entry and Control, 2007, ISBN 978-91-85831-77-7.

No 1110 **He Tan:** Aligning Biomedical Ontologies, 2007, ISBN 978-91-85831-56-2.

No 1112 **Jessica Lindblom:** Minding the body - Interacting socially through embodied action, 2007, ISBN 978-91-85831-48-7.

No 1113 **Pontus Wärnestål:** Dialogue Behavior Management in Conversational Recommender Systems, 2007, ISBN 978-91-85831-47-0.

No 1120 **Thomas Gustafsson:** Management of Real-Time Data Consistency and Transient Overloads in Embedded Systems, 2007, ISBN 978-91-85831-33-3.

No 1127 **Alexandru Andrei:** Energy Efficient and Predictable Design of Real-time Embedded Systems, 2007, ISBN 978-91-85831-06-7.

No 1139 **Per Wikberg:** Eliciting Knowledge from Experts in Modeling of Complex Systems: Managing Variation and Interactions, 2007, ISBN 978-91-85895-66-3.

No 1143 **Mehdi Amirijoo:** QoS Control of Real-Time Data Services under Uncertain Workload, 2007, ISBN 978-91-85895-49-6.

No 1150 **Sanny Syberfeldt:** Optimistic Replication with Forward Conflict Resolution in Distributed Real-Time Databases, 2007, ISBN 978-91-85895-27-4.

No 1155 **Beatrice Alenljung:** Envisioning a Future Decision Support System for Requirements Engineering - A Holistic and Human-centred Perspective, 2008, ISBN 978-91-85895-11-3.

No 1156 **Artur Wilk:** Types for XML with Application to Xcerpt, 2008, ISBN 978-91-85895-08-3.

No 1183 **Adrian Pop:** Integrated Model-Driven Development Environments for Equation-Based Object-Oriented Languages, 2008, ISBN 978-91-7393-895-2.

No 1185 **Jörgen Skågeby:** Gifting Technologies - Ethnographic Studies of End-users and Social Media Sharing, 2008, ISBN 978-91-7393-892-1.

No 1187 **Imad-Eldin Ali Abugessaisa:** Analytical tools and information-sharing methods supporting road safety organizations, 2008, ISBN 978-91-7393-887-7.

No 1204 **H. Joe Steinhauer:** A Representation Scheme for Description and Reconstruction of Object Configurations Based on Qualitative Relations, 2008, ISBN 978-91-7393-823-5.

No 1222 **Anders Larsson:** Test Optimization for Core-based System-on-Chip, 2008, ISBN 978-91-7393-768-9.

No 1238 **Andreas Borg:** Processes and Models for Capacity Requirements in Telecommunication Systems, 2009, ISBN 978-91-7393-700-9.

No 1240 **Fredrik Heintz:** DyKnow: A Stream-Based Knowledge Processing Middleware Framework, 2009, ISBN 978-91-7393-696-5.

No 1241 **Birgitta Lindström:** Testability of Dynamic Real-Time Systems, 2009, ISBN 978-91-7393-695-8.

No 1244 **Eva Blomqvist:** Semi-automatic Ontology Construction based on Patterns, 2009, ISBN 978-91-7393-683-5.

No 1249 **Rogier Woltjer:** Functional Modeling of Constraint Management in Aviation Safety and Command and Control, 2009, ISBN 978-91-7393-659-0.

No 1260 **Gianpaolo Conte:** Vision-Based Localization and Guidance for Unmanned Aerial Vehicles, 2009, ISBN 978-91-7393-603-3.

No 1262 **AnnMarie Ericsson:** Enabling Tool Support for Formal Analysis of ECA Rules, 2009, ISBN 978-91-7393-598-2.

No 1266 **Jiri Trnka:** Exploring Tactical Command and Control: A Role-Playing Simulation Approach, 2009, ISBN 978-91-7393-571-5.

No 1268 **Bahlol Rahimi:** Supporting Collaborative Work through ICT - How End-users Think of and Adopt Integrated Health Information Systems, 2009, ISBN 978-91-7393-550-0.

No 1274 **Fredrik Kuivinen:** Algorithms and Hardness Results for Some Valued CSPs, 2009, ISBN 978-91-7393-525-8.

No 1281 **Gunnar Mathiason:** Virtual Full Replication for Scalable Distributed Real-Time Databases, 2009, ISBN 978-91-7393-503-6.

No 1290 **Viacheslav Izosimov:** Scheduling and Optimization of Fault-Tolerant Distributed Embedded Systems, 2009, ISBN 978-91-7393-482-4.

No 1294 **Johan Thapper:** Aspects of a Constraint Optimisation Problem, 2010, ISBN 978-91-7393-464-0.

No 1306 **Susanna Nilsson:** Augmentation in the Wild: User Centered Development and Evaluation of Augmented Reality Applications, 2010, ISBN 978-91-7393-416-9.

No 1313 **Christer Thörn:** On the Quality of Feature Models, 2010, ISBN 978-91-7393-394-0.

No 1321 **Zhiyuan He:** Temperature Aware and Defect-Probability Driven Test Scheduling for System-on-Chip, 2010, ISBN 978-91-7393-378-0.

No 1333 **David Broman:** Meta-Languages and Semantics for Equation-Based Modeling and Simulation, 2010, ISBN 978-91-7393-335-3.

No 1337 **Alexander Siemers:** Contributions to Modelling and Visualisation of Multibody Systems Simulations with Detailed Contact Analysis, 2010, ISBN 978-91-7393-317-9.

No 1354 **Mikael Asplund:** Disconnected Discoveries: Availability Studies in Partitioned Networks, 2010, ISBN 978-91-7393-278-3.

No 1359 **Jana Rambusch**: Mind Games Extended: Understanding Gameplay as Situated Activity, 2010, ISBN 978-91-7393-252-3.

No 1373 **Sonia Sangari**: Head Movement Correlates to Focus Assignment in Swedish, 2011, ISBN 978-91-7393-154-0.

No 1374 **Jan-Erik Källhammer**: Using False Alarms when Developing Automotive Active Safety Systems, 2011, ISBN 978-91-7393-153-3.

No 1375 **Mattias Eriksson**: Integrated Code Generation, 2011, ISBN 978-91-7393-147-2.

No 1381 **Ola Leifler**: Affordances and Constraints of Intelligent Decision Support for Military Command and Control – Three Case Studies of Support Systems, 2011, ISBN 978-91-7393-133-5.

No 1386 **Soheil Samii**: Quality-Driven Synthesis and Optimization of Embedded Control Systems, 2011, ISBN 978-91-7393-102-1.

No 1419 **Erik Kuiper**: Geographic Routing in Intermittently-connected Mobile Ad Hoc Networks: Algorithms and Performance Models, 2012, ISBN 978-91-7519-981-8.

No 1451 **Sara Stymne**: Text Harmonization Strategies for Phrase-Based Statistical Machine Translation, 2012, ISBN 978-91-7519-887-3.

No 1455 **Alberto Montebelli**: Modeling the Role of Energy Management in Embodied Cognition, 2012, ISBN 978-91-7519-882-8.

No 1465 **Mohammad Saifullah**: Biologically-Based Interactive Neural Network Models for Visual Attention and Object Recognition, 2012, ISBN 978-91-7519-838-5.

No 1490 **Tomas Bengtsson**: Testing and Logic Optimization Techniques for Systems on Chip, 2012, ISBN 978-91-7519-742-5.

No 1481 **David Byers**: Improving Software Security by Preventing Known Vulnerabilities, 2012, ISBN 978-91-7519-784-5.

No 1496 **Tommy Färnqvist**: Exploiting Structure in CSP-related Problems, 2013, ISBN 978-91-7519-711-1.

No 1503 **John Wilander**: Contributions to Specification, Implementation, and Execution of Secure Software, 2013, ISBN 978-91-7519-681-7.

No 1506 **Magnus Ingmarsson**: Creating and Enabling the Useful Service Discovery Experience, 2013, ISBN 978-91-7519-662-6.

No 1547 **Wladimir Schamai**: Model-Based Verification of Dynamic System Behavior against Requirements: Method, Language, and Tool, 2013, ISBN 978-91-7519-505-6.

No 1551 **Henrik Svensson**: Simulations, 2013, ISBN 978-91-7519-491-2.

No 1559 **Sergiu Rafiliu**: Stability of Adaptive Distributed Real-Time Systems with Dynamic Resource Management, 2013, ISBN 978-91-7519-471-4.

No 1581 **Usman Dastgeer**: Performance-aware Component Composition for GPU-based Systems, 2014, ISBN 978-91-7519-383-0.

No 1602 **Cai Li**: Reinforcement Learning of Locomotion based on Central Pattern Generators, 2014, ISBN 978-91-7519-313-7.

No 1652 **Roland Samlaus**: An Integrated Development Environment with Enhanced Domain-Specific Interactive Model Validation, 2015, ISBN 978-91-7519-090-7.

No 1663 **Hannes Uppman**: On Some Combinatorial Optimization Problems: Algorithms and Complexity, 2015, ISBN 978-91-7519-072-3.

No 1664 **Martin Sjölund**: Tools and Methods for Analysis, Debugging, and Performance Improvement of Equation-Based Models, 2015, ISBN 978-91-7519-071-6.

No 1666 **Kristian Stavåker**: Contributions to Simulation of Modelica Models on Data-Parallel Multi-Core Architectures, 2015, ISBN 978-91-7519-068-6.

No 1680 **Adrian Lifa**: Hardware/Software Codesign of Embedded Systems with Reconfigurable and Heterogeneous Platforms, 2015, ISBN 978-91-7519-040-2.

No 1685 **Bogdan Tanasa**: Timing Analysis of Distributed Embedded Systems with Stochastic Workload and Reliability Constraints, 2015, ISBN 978-91-7519-022-8.

No 1691 **Håkan Warnquist**: Troubleshooting Trucks – Automated Planning and Diagnosis, 2015, ISBN 978-91-7685-993-3.

No 1702 **Nima Aghaee**: Thermal Issues in Testing of Advanced Systems on Chip, 2015, ISBN 978-91-7685-949-0.

No 1715 **Maria Vasilevskaya**: Security in Embedded Systems: A Model-Based Approach with Risk Metrics, 2015, ISBN 978-91-7685-917-9.

No 1729 **Ke Jiang**: Security-Driven Design of Real-Time Embedded System, 2016, ISBN 978-91-7685-884-4.

No 1733 **Victor Lagerkvist**: Strong Partial Clones and the Complexity of Constraint Satisfaction Problems: Limitations and Applications, 2016, ISBN 978-91-7685-856-1.

No 1734 **Chandan Roy**: An Informed System Development Approach to Tropical Cyclone Track and Intensity Forecasting, 2016, ISBN 978-91-7685-854-7.

No 1746 **Amir Aminifar**: Analysis, Design, and Optimization of Embedded Control Systems, 2016, ISBN 978-91-7685-826-4.

No 1747 **Ekhiotz Vergara**: Energy Modelling and Fairness for Efficient Mobile Communication, 2016, ISBN 978-91-7685-822-6.

No 1748 **Dag Sonntag**: Chain Graphs – Interpretations, Expressiveness and Learning Algorithms, 2016, ISBN 978-91-7685-818-9.

No 1768 **Anna Vapen**: Web Authentication using Third-Parties in Untrusted Environments, 2016, ISBN 978-91-7685-753-3.

No 1778 **Magnus Jandinger**: On a Need to Know Basis: A Conceptual and Methodological Framework for Modelling and Analysis of Information Demand in an Enterprise Context, 2016, ISBN 978-91-7685-713-7.

No 1798 **Rahul Hiran**: Collaborative Network Security: Targeting Wide-area Routing and Edge-network Attacks, 2016, ISBN 978-91-7685-662-8.

No 1813 **Nicolas Melot**: Algorithms and Framework for Energy Efficient Parallel Stream Computing on Many-Core Architectures, 2016, ISBN 978-91-7685-623-9.

No 1823 **Amy Rankin**: Making Sense of Adaptations: Resilience in High-Risk Work, 2017, ISBN 978-91-7685-596-6.

No 1831 **Lisa Malmberg**: Building Design Capability in the Public Sector: Expanding the Horizons of Development, 2017, ISBN 978-91-7685-585-0.

No 1851 **Marcus Bendtsen**: Gated Bayesian Networks, 2017, ISBN 978-91-7685-525-6.

No 1852 **Zlatan Dragisic**: Completion of Ontologies and Ontology Networks, 2017, ISBN 978-91-7685-522-5.

No 1854 **Meysam Aghighi**: Computational Complexity of some Optimization Problems in Planning, 2017, ISBN 978-91-7685-519-5.

No 1863 **Simon Ståhlberg**: Methods for Detecting Unsolvable Planning Instances using Variable Projection, 2017, ISBN 978-91-7685-498-3.

No 1879 **Karl Hammar**: Content Ontology Design Patterns: Qualities, Methods, and Tools, 2017, ISBN 978-91-7685-454-9.

No 1887 **Ivan Ukhov**: System-Level Analysis and Design under Uncertainty, 2017, ISBN 978-91-7685-426-6.

No 1891 **Valentina Ivanova**: Fostering User Involvement in Ontology Alignment and Alignment Evaluation, 2017, ISBN 978-91-7685-403-7.

No 1902 **Vengatanathan Krishnamoorthi**: Efficient HTTP-based Adaptive Streaming of Linear and Interactive Videos, 2018, ISBN 978-91-7685-371-9.

No 1903 **Lu Li**: Programming Abstractions and Optimization Techniques for GPU-based Heterogeneous Systems, 2018, ISBN 978-91-7685-370-2.

No 1913 **Jonas Rybing**: Studying Simulations with Distributed Cognition, 2018, ISBN 978-91-7685-348-1.

No 1936 **Leif Jonsson**: Machine Learning-Based Bug Handling in Large-Scale Software Development, 2018, ISBN 978-91-7685-306-1.

No 1964 **Arian Maghazeh**: System-Level Design of GPU-Based Embedded Systems, 2018, ISBN 978-91-7685-175-3.

No 1967 **Mahder Gebremedhin**: Automatic and Explicit Parallelization Approaches for Equation Based Mathematical Modeling and Simulation, 2019, ISBN 978-91-7685-163-0.

No 1984 **Anders Andersson**: Distributed Moving Base Driving Simulators – Technology, Performance, and Requirements, 2019, ISBN 978-91-7685-090-9.

No 1993 **Ulf Kargén**: Scalable Dynamic Analysis of Binary Code, 2019, ISBN 978-91-7685-049-7.

No 2001 **Tim Overkamp**: How Service Ideas Are Implemented: Ways of Framing and Addressing Service Transformation, 2019, ISBN 978-91-7685-025-1.

No 2006 **Daniel de Leng**: Robust Stream Reasoning Under Uncertainty, 2019, ISBN 978-91-7685-013-8.

No 2048 **Biman Roy**: Applications of Partial Polymorphisms in (Fine-Grained) Complexity of Constraint Satisfaction Problems, 2020, ISBN 978-91-7929-898-2.

No 2051 **Olov Andersson**: Learning to Make Safe Real-Time Decisions Under Uncertainty for Autonomous Robots, 2020, ISBN 978-91-7929-889-0.

No 2065 **Vanessa Rodrigues**: Designing for Resilience: Navigating Change in Service Systems, 2020, ISBN 978-91-7929-867-8.

No 2082 **Robin Kurtz**: Contributions to Semantic Dependency Parsing: Search, Learning, and Application, 2020, ISBN 978-91-7929-822-7.

No 2108 **Shanai Ardi**: Vulnerability and Risk Analysis Methods and Application in Large Scale Development of Secure Systems, 2021, ISBN 978-91-7929-744-2.

No 2125 **Zeinab Ganjei**: Parameterized Verification of Synchronized Concurrent Programs, 2021, ISBN 978-91-7929-697-1.

No 2153 **Robin Keskisärkkä**: Complex Event Processing under Uncertainty in RDF Stream Processing, 2021, ISBN 978-91-7929-621-6.

No 2168 **Rouhollah Mahfouzi**: Security-Aware Design of Cyber-Physical Systems for Control Applications, 2021, ISBN 978-91-7929-021-4.

No 2205 **August Ernstsson**: Pattern-based Programming Abstractions for Heterogeneous Parallel Computing, 2022, ISBN 978-91-7929-195-2.

No 2218 **Huanyu Li**: Ontology-Driven Data Access and Data Integration with an Application in the Materials Design Domain, 2022, ISBN 978-91-7929-267-6.

No 2219 **Evelina Rennes**: Automatic Adaption of Swedish Text for Increased Inclusion, 2022, ISBN 978-91-7929-269-0.

No 2220 **Yuanbin Zhou**: Synthesis of Safety-Critical Real-Time Systems, 2022, ISBN 978-91-7929-271-3.

No 2247 **Azeem Ahmad**: Contributions to Improving Feedback and Trust in Automated Testing and Continuous Integration and Delivery, 2022, ISBN 978-91-7929-422-9.

No 2248 **Ana Kuštrak Korper**: Innovating Innovation: Understanding the Role of Service Design in Service Innovation, 2022, ISBN 978-91-7929-424-3.

Linköping Studies in Arts and Sciences

No 504 **Ing-Marie Jonsson:** Social and Emotional Characteristics of Speech-based In-Vehicle Information Systems: Impact on Attitude and Driving Behaviour, 2009, ISBN 978-91-7393-478-7.

No 586 **Fabian Segelström:** Stakeholder Engagement for Service Design: How service designers identify and communicate insights, 2013, ISBN 978-91-7519-554-4.

No 618 **Johan Blomkvist:** Representing Future Situations of Service: Prototyping in Service Design, 2014, ISBN 978-91-7519-343-4.

No 620 **Marcus Mast:** Human-Robot Interaction for Semi-Autonomous Assistive Robots, 2014, ISBN 978-91-7519-319-9.

No 677 **Peter Berggren:** Assessing Shared Strategic Understanding, 2016, ISBN 978-91-7685-786-1.

No 695 **Mattias Forsblad:** Distributed cognition in home environments: The prospective memory and cognitive practices of older adults, 2016, ISBN 978-91-7685-686-4.

No 787 **Sara Nygårdhs:** Adaptive behaviour in traffic: An individual road user perspective, 2020, ISBN 978-91-7929-857-9.

No 811 **Sam Thellman:** Social Robots as Intentional Agents, 2021, ISBN, 978-91-7929-008-5.

Linköping Studies in Statistics

No 9 **Davood Shahsavani:** Computer Experiments Designed to Explore and Approximate Complex Deterministic Models, 2008, ISBN 978-91-7393-976-8.

No 10 **Karl Wahlin:** Roadmap for Trend Detection and Assessment of Data Quality, 2008, ISBN 978-91-7393-792-4.

No 11 **Oleg Sysoev:** Monotonic regression for large multivariate datasets, 2010, ISBN 978-91-7393-412-1.

No 13 **Agné Burauskaite-Harju:** Characterizing Temporal Change and Inter-Site Correlations in Daily and Subdaily Precipitation Extremes, 2011, ISBN 978-91-7393-110-6.

No 14 **Måns Magnusson:** Scalable and Efficient Probabilistic Topic Model Inference for Textual Data, 2018, ISBN 978-91-7685-288-0.

No 15 **Per Sidén:** Scalable Bayesian spatial analysis with Gaussian Markov random fields, 2020, 978-91-7929-818-0.

Linköping Studies in Information Science

No 1 **Karin Axelsson:** Metodisk systemstrukturering- att skapa samstämmighet mellan informationssystemarkitektur och verksamhet, 1998. ISBN 9172-19-296-8.

No 2 **Stefan Cronholm:** Metodverktyg och användbarhet - en studie av datorstödd metodbaserad systemutveckling, 1998, ISBN 9172-19-299-2.

No 3 **Anders Avdic:** Användare och utvecklare - om anveckling med kalkylprogram, 1999. ISBN 91-7219-606-8.

No 4 **Owen Eriksson:** Kommunikationskvalitet hos informationssystem och affärsprocesser, 2000, ISBN 91-7219-811-7.

No 5 **Mikael Lind:** Från system till process - kriterier för processbestämning vid verksamhetsanalys, 2001, ISBN 91-7373-067-X.

No 6 **Ulf Melin:** Koordination och informationssystem i företag och nätverk, 2002, ISBN 91-7373-278-8.

No 7 **Pär J. Ågerfalk:** Information Systems Actability - Understanding Information Technology as a Tool for Business Action and Communication, 2003, ISBN 91-7373-628-7.

No 8 **Ulf Seigerroth:** Att förstå och förändra systemutvecklingsverksamheter - en taxonomi för metautveckling, 2003, ISBN 91-7373-736-4.

No 9 **Karin Hedström:** Spår av datoriseringens värden – Effekter av IT i äldreomsorg, 2004, ISBN 91-7373-963-4.

No 10 **Ewa Braf:** Knowledge Demanded for Action - Studies on Knowledge Mediation in Organisations, 2004, ISBN 91-85295-47-7.

No 11 **Fredrik Karlsson:** Method Configuration method and computerized tool support, 2005, ISBN 91-85297-48-8.

No 12 **Malin Nordström:** Styrbar systemförvaltning - Att organisera systemförvaltningsverksamhet med hjälp av effektiva förvaltningsobjekt, 2005, ISBN 91-85297-60-7.

No 13 **Stefan Holgersson:** Yrke: POLIS - Yrkeskunskap, motivation, IT-system och andra förutsättningar för polisarbete, 2005, ISBN 91-85299-43-X.

No 14 **Benneth Christiansson, Marie-Therese Christiansson:** Mötet mellan process och komponent - mot ett ramverk för en verksamhetsnära kravspecifikation vid anskaffning av komponentbaserade informationssystem, 2006, ISBN 91-85643-22-X.

FACULTY OF SCIENCE AND ENGINEERING

Linköping Studies in Science and Technology, Dissertation No. 2247, 2022
Department of Computer and Information Science

Linköping University
SE-581 83 Linköping, Sweden

www.liu.se