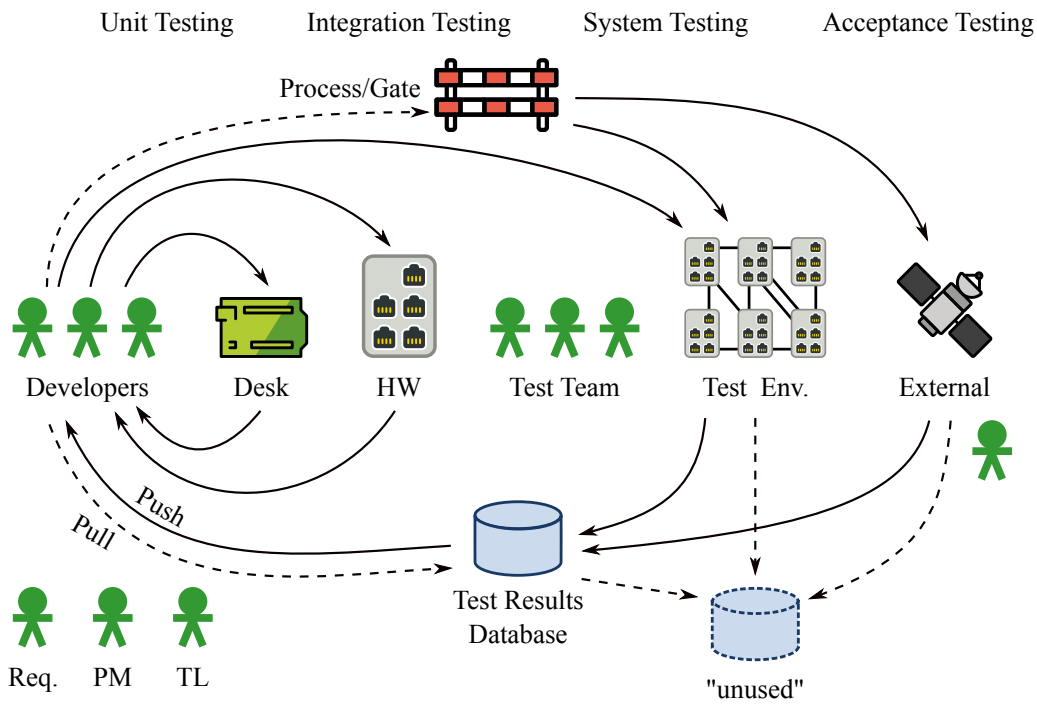
Mälardalen University Doctoral Dissertation 349

Automated System-Level Software Testing of Industrial Networked Embedded Systems

Per Erik Strandberg



MÄLARDALEN UNIVERSITY
SWEDEN

Mälardalen University Press Dissertations
No. 349

AUTOMATED SYSTEM-LEVEL SOFTWARE TESTING OF INDUSTRIAL NETWORKED EMBEDDED SYSTEMS

Per Erik Strandberg

2021

School of Innovation, Design and Engineering

ISBN 978-91-7485-529-6
ISSN 1651-4238
Printed by E-Print AB, Stockholm, Sweden

Mälardalen University Press Dissertations
No. 349

AUTOMATED SYSTEM-LEVEL SOFTWARE TESTING OF INDUSTRIAL NETWORKED EMBEDDED SYSTEMS

Per Erik Strandberg

Akademisk avhandling

som för avläggande av teknologie doktorsexamen i datavetenskap vid Akademin för innovation, design och teknik kommer att offentligen försvaras måndagen den 22 november 2021, 13.15 i Gamma och online via Teams, Mälardalens högskola, Västerås.

Fakultetsopponent: Professor Burak Turhan, University of Oulu

Akademin för innovation, design och teknik

Abstract

Embedded systems are ubiquitous and play critical roles in management systems for industry and transport. Software failures in these domains may lead to loss of production or even loss of life, so the software in these systems needs to be reliable. Software testing is a standard approach for quality assurance of embedded software, and many software development processes strive for test automation. Out of the many challenges for successful software test automation, this thesis addresses five: (i) understanding how updated software reaches a test environment, how testing is conducted in the test environment, and how test results reach the developers that updated the software in the first place; (ii) selecting which test cases to execute in a test suite given constraints on available time and test systems; (iii) given that the test cases an run on different configurations of connected devices, selecting which hardware to use for each test case to be executed; (iv) analyzing test cases that, when executed over time on evolving software, testware or hardware revisions, appear to randomly fail; and (v) making test results information actionable with test results exploration and visualization.

The challenges are tackled in several ways. First, to better understand the flow of information in the embedded systems software development process, interviews at five different companies were conducted. The results show how visualizations and a test results database support decision-making. Results also describe the overall flow of information in software testing: from developers to hardware in the test environment, and back to developers. Second, in order to address the challenges of test selection and hardware selection, automated approaches for testing given resource constraints were implemented and evaluated using industrial data stemming from years of nightly testing. It was shown that these approaches could solve problems such as nightly testing not finishing on time, as well as increasing hardware coverage by varying hardware selection over test iterations. Third, the challenge of intermittently failing tests was addressed with a new metric that can classify test cases as intermittently or consistently failing. Again, by using industry data, factors that lead to intermittent failures were identified, and similarities and differences between root causes for intermittently and consistently failing tests were observed. Finally, in order to better render test results actionable, a tool was implemented for test results exploration and visualization. The implementation was evaluated using a reference group and logging of the tool's usage. Solution patterns and views of the tool were identified, as well as challenges for implementing such a tool.

ISBN 978-91-7485-529-6
ISSN 1651-4238

And into the forest I go to lose my mind and find my soul.

– John Muir (1838-1914)

Graphical Abstract

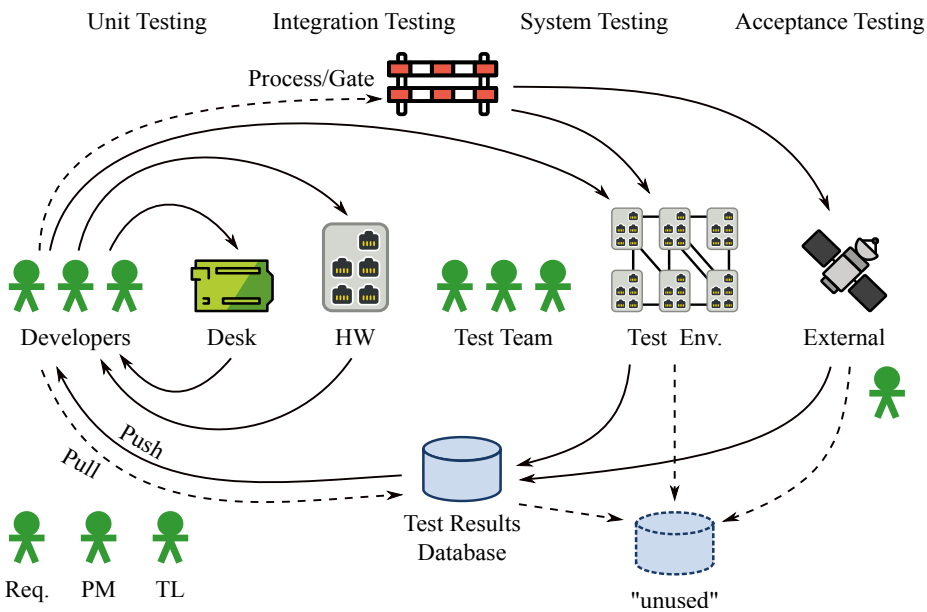


Figure 1: Automated system-level software testing of industrial networked embedded systems.

In the software development of embedded systems, one has to test the software as it runs on physical devices. The testing can be automated, and this thesis covers some of the challenges that can arrive: How does the information related to testing flow? How should test cases and hardware devices be selected for the testing? Why do some test cases change verdict for no apparent reason? Finally, how can test results be made explorable and visualized?

Abstract

Embedded systems are ubiquitous and play critical roles in management systems for industry and transport. Software failures in these domains may lead to loss of production or even loss of life, so the software in these systems needs to be reliable. Software testing is a standard approach for quality assurance of embedded software, and many software development processes strive for test automation. Out of the many challenges for successful software test automation, this thesis addresses five: (i) understanding how updated software reaches a test environment, how testing is conducted in the test environment, and how test results reach the developers that updated the software in the first place; (ii) selecting which test cases to execute in a test suite given constraints on available time and test systems; (iii) given that the test cases are run on different configurations of connected devices, selecting which hardware to use for each test case to be executed; (iv) analyzing test cases that, when executed over time on evolving software, testware or hardware revisions, appear to randomly fail; and (v) making test results information actionable with test results exploration and visualization.

The challenges are tackled in several ways. First, to better understand the flow of information in the embedded systems software development process, interviews at five different companies were conducted. The results show how visualizations and a test results database support decision-making. Results also describe the overall flow of information in software testing: from developers to hardware in the test environment, and back to developers. Second, in order to address the challenges of test selection and hardware selection, automated approaches for testing given resource constraints were implemented and evaluated using industrial data stemming from years of nightly testing. It was shown that these approaches could solve problems such as nightly testing not finishing on time, as well as increasing hardware coverage by varying hardware selection over test iterations. Third, the challenge of intermittently failing tests was addressed with a new metric that can classify test cases as intermittently or consistently failing. Again, by using industry data, factors that lead to inter-

mittent failures were identified, and similarities and differences between root causes for intermittently and consistently failing tests were observed. Finally, in order to better render test results actionable, a tool was implemented for test results exploration and visualization. The implementation was evaluated using a reference group and logging of the tool's usage. Solution patterns and views of the tool were identified, as well as challenges for implementing such a tool.

Sammanfattning

Inbyggda system finns överallt och fyller viktiga roller i ledningssystem för industri och transport. Här kan mjukvarufel leda till produktionsbortfall eller till och med dödsfall, så mjukvaran i systemen måste vara tillförlitlig. Mjukvarutestning är en standardmetod för att kvalitetssäkra mjukvaran i inbyggda system, och många processer för mjukvaruutveckling strävar efter automatiserad testning. Av de många utmaningarna för framgångsrik testautomatisering täcker denna avhandling fem: (i) att förstå hur uppdaterad mjukvara når en testmiljö, hur testning utförs i testmiljön och hur testresultat når tillbaka till utvecklarna som uppdaterade mjukvaran; (ii) att välja vilka testfall som ska exekveras i en testsvit givet begränsningar i tillgänglig tid och tillgängliga testsystem; (iii) givet att testfall körs på olika konfigurationer av anslutna enheter, hur väljs hårdvaran som ska användas för varje testfall ut; (iv) att analysera testfall som, när de körs upprepade gånger med mjukvara, testvara eller hårdvara under utveckling, verkar slumpmässigt påvisa fel; och (v) att göra testresultat mer användbara genom göra det möjligt att utforska och visualisera den.

Utmaningarna möts på flera sätt. För det första, för att bättre förstå informationsflödet i utvecklingsprocessen för mjukvaran i inbyggda system så genomfördes intervjuer på fem olika företag. Resultaten visar hur visualiseringar och en testresultatsdatabas stödjer beslutsfattande. Resultaten beskriver också det övergripande informationsflödet i mjukvarutestning: från utvecklare till hårdvara i testmiljön och tillbaka till utvecklarna. För det andra, för att ta itu med utmaningarna vid testselektion och hårdvaruselektion, implementerades och utvärderades automatiserade metoder för testning givet resursbegränsningar, med hjälp av industriella data från flera år av nattliga tester. Dessa tillvägagångssätt kunde lösa problem som att nattliga tester inte avslutas i tid, och kan även öka hårdvarutäckningen genom att variera hårdvaruselektionen över testiterationer. För det tredje angrips utmaningen med intermittenta tester med en ny metrik för att klassificera testfall som intermittent eller konsekvent påvisar fel. Genom att återigen använda

industriellt data så identifierades faktorer som leder till intermittenta tester, och likheter och skillnader i rotorsaker för intermittenta och konsekventa tester identifierades. Slutligen, för att göra testresultaten mer användbara, implementerades ett verktyg för utforskning och visualisering av testresultat. Implementationen utvärderades med hjälp av en referensgrupp samt av loggning av verktygets användning. Lösningsmönster och verktygets vyer identifierades, liksom utmaningar för att implementera ett sådant verktyg.

Populärvetenskaplig sammanfattning

I moderna industriella automatiseringssystem, som ombord på tåg eller i elkraftsystem, spelar kommunikationsnätverket en kritisk roll. Bortfall av service kan få allvarliga konsekvenser som minskad produktion eller eventuellt dödsfall. Mjukvaran i inbyggda system i nätverk måste vara robust, och mjukvarutestning är en standardmetod för kvalitetskontroll. Denna testning är dyr, repetitiv, kan begränsa tiden till marknad och drabbas ofta av förseningar. Testautomatisering är därför önskvärd, men kan komma med egna utmaningar.

Westermo Network Technologies AB har samarbetat i ett forskningsprojekt med Mälardalens Högskola där Per Erik Strandberg undersökt testautomatisering av dessa system. Strandbergs forskning har fokuserat på flera utmaningar. För det första, hur går det till när mjukvara skapas av en programmerare och ska nå en testmiljö? Vilka processer och hinder spelar roll för ett bra informationsflöde i den processen? För det andra, hur testar vi på ett smartare sätt givet begränsade resurser: ska vi alltid köra alla testfall? Vilken hårdvara ska vi välja för testningen? För det tredje, varför är det så att vissa tester liksom "byter åsikt" oftare än andra – ett test borde väl helt konsekvent alltid peka ut ett fel när felet finns där, och aldrig när felet inte finns? Slutligen, hur kan vi använda den ökande mängd data om testresultat som kommer från ökad testautomatisering? Hur kan resultaten visualiseras och hur gör vi det möjligt att utforska den?

En del av Strandbergs forskning är inriktad på förbättrade algoritmer och implementering av dessa i verktyg. Ett exempel på ett sådant verktyg är Svitbyggaren som väljer de viktigaste testfallen givet ett antal prioriteringskriterier, ett andra verktyg väljer hårdvara för testningen, och ett tredje pekar ut tester som oftare än andra byter åsikt. För att kunna visa att dessa verktyg utökar forskningsfronten så gjorde han utvärderingar med industriell data som samlats in under flera år. Däribland detaljer om miljoner av testexekveringar, och

även information om hur hårdvara organiserats i testsystem. Resultaten visar att förbättrade verktyg löser kritiska problem: med Svitbyggaren så avslutas nu testningen i tid och tillkortakommanden i kvalitet hittas tidigare. Vidare så kan allokeringen av hårdvara ändras över tid, något som förbättrar testtäckningen. Tester som byter åsikt ofta kan identifieras och orsakerna till varför de är intermittenta har undersökts.

En annan delen av Strandbergs forskning använder kvalitativa metoder där intervjuer med utövare är centrala. Han transkriberade mer än 28 timmar ljud till mer än 185 sidor kompakt text som sedan analyserades med metoder med ursprung i forskning om psykologi. I dessa studier visar Strandberg hur testresultat nu kan visualiseras och hur beslut tas med hjälp av en databas med testresultat. Han visar även det övergripande informationsflödet i mjukvarutestningsprocesserna, samt teman, utmaningar och bra tillvägagångssätt. De viktigaste bra tillvägagångssätten är: nära samarbete och kommunikation mellan roller. Till sist visar Strandberg på utmaningar med att visualisera testresultat, till exempel hur man ska möta användarnas förväntningar, oväntade avvikelser i testningen och hur information från olika system ska integreras i ett.

Det finns ett gap mellan industri och akademi inom fältet mjukvarutestning. Strandbergs resultat och beskrivningar av verktyg kan guida och inspirera andra industriella utövare. Det är uppenbart att framtida forskning om mjukvarutestning skulle dra nytta av ett fortsatt samarbete mellan industri och akademi, och Strandberg hoppas spela en fortsatt roll i överbryggandet av gapet.

Acknowledgments

By writing a doctoral thesis when 40^{+}, I am old enough to have had a my life influenced by a lot of people. Many are those who have helped me move towards research, many have helped me believe in myself, and in this research.

First of all, I would like to thank my family: my mother Gudrun for opening the door to the wonderful world of mathematics; my father Jan for teaching me to see the positive things in life and, in particular, to not care so much about the negative; my brother Jonas for helping me have two feet on the ground; my wife Anna for always being there for me; and especially my children Knut and Vera for helping me rediscover the magic of playing, for teaching me about emotions, and for being the stochastic process that regularly adds chaos into my otherwise gray life.

It has been said that "when the student is ready the teacher will appear."[1] For me, this was almost literally the case. I would like to thank Raimo for first suggesting industrial doctoral studies all those years ago, and to Monika and Kristian for pushing me in the right direction. Before I knew it, teachers had appeared – thank you Daniel and Wasif for supervising me, and thank you Tom, Elaine, Eddie, Robert, Gita and Jonathan for co-authoring with me.

My research has not only included an academic partner, but also an industrial side – there are many awesome managers who made this research possible or that have had a positive impact in my life: thank you Patric, Peter, Petra, Pierre, and Peter. I would also like to thank all colleagues, past and present. In particular, thanks to the testing competence group at HiQ, and to the "Delta team" at Westermo.

I would like to thank all the wonderful people at MDH, in particular: Carola, Jenny and Malin, for ruling with an iron fist in a silk glove; Maria, Mats and Susanne, for managing the ITS ESS-H research school; Ann-Louise, Atieh, Daniel, Inna, Johan, Mahshid, Martin, Niclas, Xu and Yurii, for always

[1]The saying is sometimes falsely attributed to Lao Tzu (6th-century BCE), but seems to originate from the 1885 book *Light on the Path*, by Mabel Collins, according to Stefan Stenudd's 2020 book *Fake Lao Tzu Quotes: Erroneous Tao Te Ching Citations Examined*.

having interesting discussions in the research school; Caroline for the introduction to qualitative interview data analysis; Adnan and Aida, for always having a constructive attitude; and to Adnan, Daniel, Daniel and Jean, in the Software Testing Laboratory research group, for never backing out of a technical discussion, no matter how deep the rabbit hole would go.

To everyone I cited: thank you for letting me stand on your shoulders. To every named or anonymous reviewer: thank you for the excellent feedback and for making my research better. Thanks to the university library, and everyone else that funds open access publications.

Thanks to all the incredible people behind the many free and also commercial software projects and tools I relied on for making this thesis, in particular: Bulma, Docker, Dropbox, DuckDuckGo, Ecosia, Emacs, Firefox, FontAwesome, GNU, Gimp, git, golang, Google, Inkscape, LaTeX, Linux, MariaDB, Matplotlib, Microsoft, MySQL, OpenMoji, Overleaf, Python, SQLite, Tango, Trello, Ubuntu, Vue, and Xubuntu.

Thank you John Cleese, for all things completely different.

Finally, to you, the reader: thank you, and may you live long and prosper.

My research was funded by Westermo and the Swedish Knowledge Foundation through grants 20150277 (ITS ESS-H), and 20160139 (TESTMINE).

The saying that "when the student is ready the teacher will appear," has a second part: "when the student is truly ready the teacher will disappear." Now that my doctoral studies are almost completed, I worry that the teachers will disappear. Of course they will, some day. I hope that day will not be too soon, that perhaps other teachers will come, and that I will cross paths with wonderful teachers, coauthors, managers, and colleagues many more times. But first I hope to go into the forest, lose my mind, and find my soul.

Per Erik Strandberg
in Västerås, Sweden
on a rainy day in August 2021

List of Publications

This doctoral thesis is a collection of publications. These are listed below, as is peripheral work done during the doctoral studies. The included papers have been reformatted to better match the format of this thesis. The texts and images are almost identical to the ones published, only minimal corrections have been done to spelling, etc. The papers also have their own bibliographies. In addition, the first part of this thesis has a separate bibliography.

I was the main author, driver and writer of the publications included in the thesis. Presentations for some papers have been recorded in order to simplify dissemination, and links to available presentations are included below.

Publications Included in this Thesis

Paper A: 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." In *IEEE Access,* 7:46434–46453, 2019 [121] (Instrument: [120]).

Presentation: https://youtu.be/KVVVxe3dH8o

Paper B: P. E. Strandberg, D. Sundmark, W. Afzal, T. J. Ostrand, and E. J. Weyuker. "Experience report: Automated System Level Regression Test Prioritization using Multiple Factors." In *International Symposium on Software Reliability Engineering*, IEEE, 2016 [125].

Winner of **best research paper** award at ISSRE'16.

Paper C: P. E. Strandberg, T. J. Ostrand, E. J. Weyuker, D. Sundmark, and W. Afzal. "Automated Test Mapping and Coverage for Network Topologies." In *International Symposium on Software Testing and Analysis*, ACM, 2018 [124].

Paper D: P. E. Strandberg, T. J. Ostrand, E. J. Weyuker, D. Sundmark, and W. Afzal. "Intermittently Failing Tests in the Embedded Systems Domain."

In *International Symposium on Software Testing and Analysis*, ACM, 2020 [123].

Presentation: https://youtu.be/W1G5hVfp_Sw

Paper E: P. E. Strandberg, W. Afzal and D. Sundmark. "Software Test Results Exploration and Visualization with Continuous Integration and Nightly Testing."

Submitted to Springer's International Journal on Software Tools for Technology Transfer (STTT) in July 2021.

Publications not Included in Thesis

Paper X1: P. E. Strandberg, W. Afzal, T. Ostrand, E. Weyuker, and D. Sundmark. Automated System Level Regression Test Prioritization in a Nutshell. *IEEE Software*, 34(4):30-37, 2017 [118].

Technical Report X2: P. E. Strandberg. "Software Test Data Visualization with Heatmaps – an Initial Survey." Technical report, Mälardalen Real-Time Research Centre, Mälardalen University, MDH-MRTC-318/2017-1-SE, 2017 [115].

Paper X3: P. E. Strandberg, W. Afzal and D. Sundmark. "Decision Making and Visualizations Based on Test Results." In *International Symposium on Empirical Software Engineering and Measurement*, ACM/IEEE, 2018 [119].

Licentiate Thesis X4: P. E. Strandberg. "Automated System Level Software Testing of Networked Embedded Systems." Licentiate Thesis, Mälardalen University, 2018 [116].

Defended at Mälardalen University with a grading committee of Mika Mäntylä (Oulu University, Finland), Helena Holmström Olsson (Malmö University, Sweden), and Andrea Arcuri (Kristiania University College, Norway).

Paper X5: P. E. Strandberg. "Ethical Interviews in Software Engineering." In *International Symposium on Empirical Software Engineering and Measurement*, ACM/IEEE, 2019 [117].

Presentation: https://youtu.be/WkukdVxc8Y0

Paper X6: P. E. Strandberg, M. Frasheri and E. P. Enoiu "Ethical AI-Powered Regression Test Selection." In *International Conference on Artificial Intelligence Testing*, IEEE, 2021 [122].

Presentation: https://youtu.be/mcYXHpuaqQ8

Contents

Part I

Thesis

Chapter 1

Introduction

This chapter starts with a short history of industry-academia collaboration and industrial doctoral studies, before it introduces the personal and industrial context of this thesis, and then it gives a background to the thesis.

1.1 Brief History of Research in Industry-Academia Collaboration and Industrial Doctoral Studies

Arora et al. [3] investigate the changes in American innovation from 1850 and onwards. In particular, they discuss the golden age of the corporate labs, with AT&T's Bell Labs that employed 15 thousand people (roughly a tenth with PhDs) in the late 1960s. Fourteen Nobel Prizes and five Turing Awards were awarded to alumni at Bell Labs. The authors mention that since the 1980s, the distances between universities and corporations have grown. Universities focused on research on smaller and smaller problems, leading to innovation that could not be used at companies without great difficulty. Corporations instead focused on development. At this time, corporate research started declining – publications per firm decreased, and patents per firm has increased (except in the life sciences). Some of the possible reasons for this decline is the difference in attitude towards research, where corporate research is often more mission-oriented, and university research sometimes curiosity-driven. More concretely, the decline might be caused by (i) spill-over - where rivals use publications from a company in their patents, (ii) narrowed firm scope, (iii) increased distances between manufacturing and R&D due to changes in trade, outsourcing, and offshoring. Finally, Arora et al. mention that (iv) tapping into knowledge and invention from external sources has been simplified.

According to Geschwind [44], the Swedish doctoral training and degree

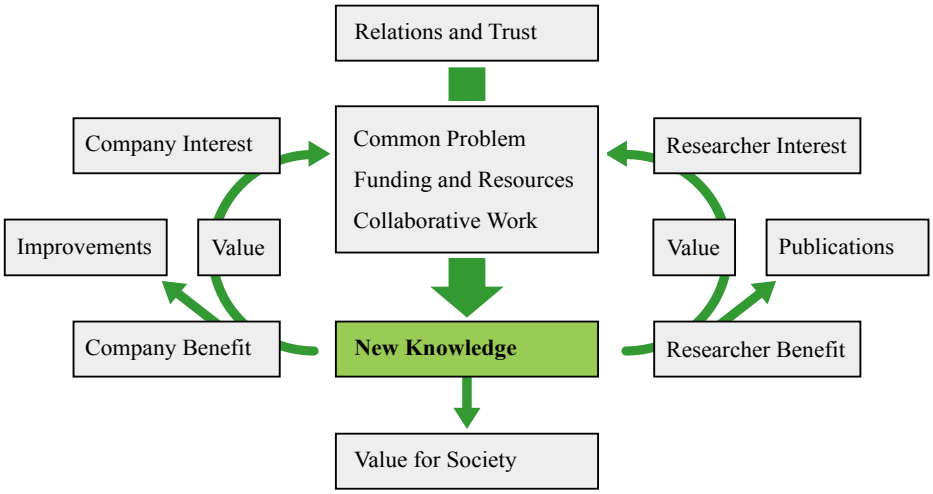


Figure 1.1: Industry-Academia collaboration based on Sannö et al. [106].

was reformed in 1969, inspired by US systems. Doctoral studies contain research and course work, corresponding to 4 years full time work. In general, since the 1960s, there has been an increased number of doctoral students, and since the end of the last millennia also an increase in throughput. The share of women has risen from 16% 1962 to about 50% in 2005. There has also been a reduced admission in humanities and about 40% of admitted doctoral students in Sweden were recruited from abroad. In 1977, a reform led to formation of several new universities in Sweden. Three reasons for the reform were to provide universal access to higher education, to limit outward migration stemming from industrial restructuring, and to provide skilled employees to industry [4]. According to Smith [110], in the mid 1990s the Swedish foundation for the advancement of knowledge and competence started offering part of funding for research if universities collaborated with industry. By 2000, thirteen industrial research schools had started and in 2017, Chalmers University of Technology alone had about 160 industrial doctoral students.

Assbring and Nuur [4] made a case study on the perceived industrial benefits of participating in collaborative research school. They found that motivators for companies include access to new knowledge and state-of-the-art research, competence creation and retention, new or improved products and processes, as well as legitimacy. Small firms are more mission-oriented, whereas larger firms are more motivated by more general goals such as developing competence.

The Swedish Higher Education Act [126] states that higher education in-

stitutions shall involve external parties and ensure that they can benefit from research findings. Doing co-production, i.e. having an industry-academia collaboration in research, is not only a way to comply with regulations and recommendations, it can also be a suitable way to conduct relevant research. For co-produced research to be *relevant* and successful, the research party and the industry party should share a common understanding of the problem and be able to communicate [31, 43, 55, 106]. This can be a challenge, as Sannö et al. point out [106], because these two parties typically have differences in perspective with respect to problem formulation, methodology, and result; as well as counterproductive differences in their views on knowledge and in their driving forces. Lo et al. [73] and Carver et al. [17] made studies in 2015 and 2016 on how practitioners perceived relevance of research in software engineering. There seems to be no correlation between citation count and its perceived relevance, and papers with industrial co-authors were only marginally more relevant. Sannö et al. discuss ways to increase the impact of industry-academia collaboration and present the model illustrated in Figure 1.1. As mentioned, the formulation of a common problem is central. The model also illustrates that the impact of the collaboration does not *end* with new knowledge – instead the new knowledge can be seen as what *drives* further industry-academia research through different benefits. Weyuker and Ostrand [134] recommend that academics strive to involve at least one industry participant that is committed to the goal of the study, and that the industry partner sees the research as relevant, valuable and important. Eldh points out that an interest in continued partnership is a good evaluation criteria for both industry and academia [31]. A more detailed approach for industry-academia collaboration was proposed by Marijan and Gotlieb [78]. Their process involves a model in seven phases starting with problem scoping (focusing on the industrial problem) and knowledge conception (to formulate research problem); followed by development, transfer, exploitation and adoption of knowledge and technology; and ending in market research in order to explore if benefits from the research could reach outside of the project. All research problems and questions in this thesis have been formulated jointly while involving both industry and academia.

There is no doubt a great value in corporate research. Arora et al. [3] argue that these labs solve practical problems, three of the reasons are that companies have the ability to test innovation at scale by having large data sets, companies are often multi-disciplinary and may have unique equipment. Furthermore, doctoral studies is no longer only perceived as a preparation for an academic career, PhD holders are attractive to industry [4]. Finally, many industrial doctoral students feel privileged to carry out research in close collaboration with industry [4].

1.2 Personal and Industrial Context

Providing information on *context* in software engineering research is relevant in order to allow readers to draw valid conclusions from the research [27, 94, 105]. In addition, in research dealing with qualitative data, the researcher is "*active* in the research process" and there is a risk for researcher bias where personal background and industrial context could play a role [14, 93]. Here I mention context such that a reader could take this into consideration when reading this thesis.

Before starting as an industrial doctoral student in 2017, I worked for 11 years with software testing, development, and requirements in the rail, nuclear, web, and communication equipment domains. During the majority of this time, I was a consultant, and as such, I led a competence network on software testing for 5 years. In this period of my life, I studied many test and requirements certification syllabi and became a certified tester as well as a certified professional for requirements engineering (ISTQB foundation, ISTQB test manager, ISTQB agile tester, and REQB CPRE Foundation).

I am employed full time at Westermo Network Technologies AB (Westermo), where I have worked with test automation, test management and research. The company designs and manufactures robust data communication devices for harsh environments, providing communication infrastructure for control and monitoring systems where consumer grade products are not sufficiently resilient. These devices and the software in them, are tested nightly with the main purpose of finding software regressions. In order to test for regressions, a number of test systems built up of physical or virtualized devices under test running the software under test have been constructed (see example in 1.2). Automated test cases have also been implemented, as well as a test framework for running the test cases. I first added code to this framework in 2011, when it was being migrated into what is now its current form. More recently, this framework has started a transition from Python into the Go programming language. Over time, more and more components have been added to this eco-system for nightly testing, such as the test selection process described in Paper B, the test case mapping onto test systems described in Paper C, and the test results database that is central to Paper E.

In 2014, when the implementation of a test selection tool was evaluated, I wrote an internal report at Westermo. At this time, I was encouraged to improve the report and publish it in an academic context by the software manager. Around the same time, I attended a course for becoming ISTQB Certified Test Manager. Through luck, connections and curiosity I got in contact with the Software Testing Laboratory at MDH. After some encouragement and coach-

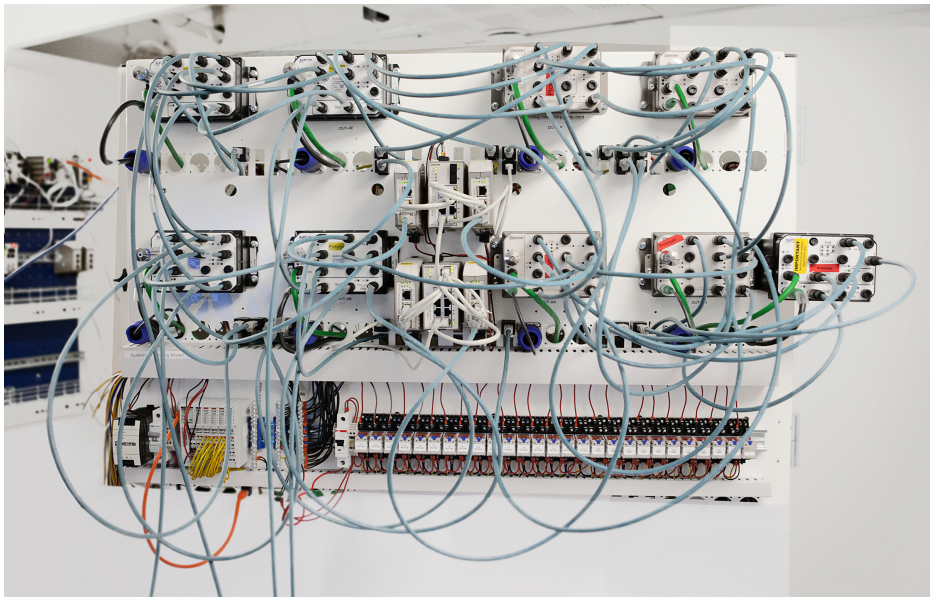


Figure 1.2: An example of a Westermo test system.

ing on how to write an academic paper, I submitted an early version of Paper B without coauthors. After being rejected with very constructive feedback, we conducted an evaluation of changes in fault distribution. Together with four other authors, I submitted to ISSRE in 2016 and the paper won the best research paper award. In parallel, I learned about the upcoming Industrial Graduate School ITS ESS-H, and after several rounds of discussion, I started as an industrial doctoral student in 2017.

During this period, Westermo migrated from a development model loosely based on scrum to a feature-driven development model based on kanban, where every feature is developed in isolation from the others in separate code branches, and then merged into a main branch after a risk management process. This increased parallelization of the testing of the branches makes it harder to get resources for nightly testing, and makes the results more sparse, in turn making the test results more complicated to understand. As illustrated in Figure 1.3, the increase in complexity is not only visible in number of branches, but also in the number of test cases and test systems in use, as well as in the number of new test cases executed.

My personal and industrial background has given me unique insights in industrial software testing, which may have had an impact on how problems have been formulated, and how data has been collected and analyzed.

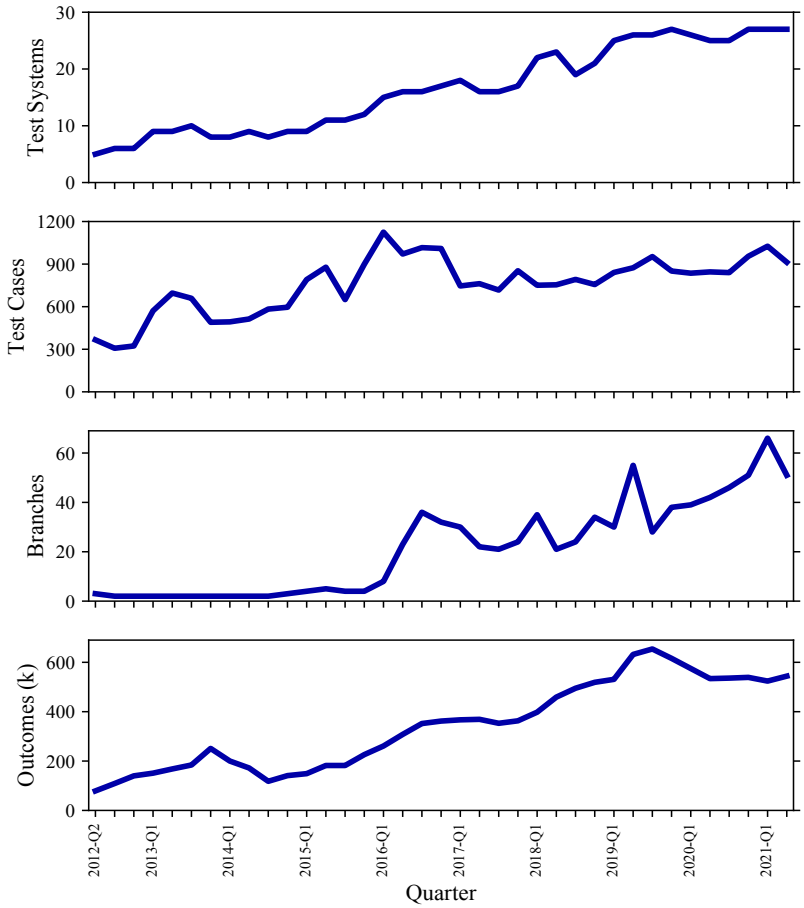


Figure 1.3: Growth in test complexity at Westermo between 2012 and Q2 2021.

1.3 Background

The growth in complexity in the nightly testing (illustrated in Figure 1.3) is also present elsewhere, such as in the automotive industry. Staron [113] reports on software architecture in this domain. There has been a growth in number of computers in a car, from at most one in the 1970s, followed by the introduction of electronic fuel injection, anti-lock brakes, cruise control in the late 1900s and autonomous driving and automated breaking when obstacles are detected in the early 2000s. Staron mentions that cars in 2015 could have 150 computers with more than 100 million lines of code, and that some of the trends here are: heterogeneity and distribution of software, variants and

configuration, autonomous functions as well as connectivity and cooperation between nodes.

Embedded systems are becoming ubiquitous. They range from portable sensors to communication equipment providing infrastructure, as part of a train or other vehicles, industrial plants or in other applications. As many as 90% of newly produced processors are part of embedded systems [41]. Software failures in communication equipment can lead to isolation of nodes in a vehicle or a plant, leading to delays, loss of productivity, or even loss of life in extreme cases. The software in embedded systems needs to be of high quality and software testing is the standard method for detecting shortcomings in quality.

Software testing, can be defined[1] as the act of manually or automatically inspecting or executing software with or without custom hardware in order to gather information for some purpose: feedback, quality control, finding issues, building trust, or other. An important aspect of testing embedded systems is to do testing on real hardware [7, 137]. For some of the testing, emulators and simulators can be suitable, but these should be complemented with physical hardware [102]. Testing could also iterate from being on host and target hardware [21]. By testing on target hardware, timing and other non-functional aspects can be verified along with functional correctness. One common approach for testing on real hardware is to build test systems of the embedded devices in network topologies, in order to support testing.

An overwhelming majority of the software testing conducted in industry is manual. Kasurinen et al. found it as high as 90% in a study in Finland in 2010 [63]. A practitioner focused report from 2015 found that only 28% of test cases are automated [111]. Test automation can reduce the cost involved and also improve time to market [135]. If tests can run with minimal human intervention, considerable gains in test efficiency are possible. Indeed, many organizations strive for agile and continuous development processes, where test automation is an important part, and research has been done on how to achieve this [89]. For organizations developing networked embedded systems to be successful in this transition, they would need a number of things in addition to agile processes. First of all, they would need a way to automate testing with a test framework and test cases. Test suites would have to run every now and then. The test framework needs to know how to log in, upgrade, and configure the hardware in the test systems. If the organizations develop many different hardware models, they would need several test systems with different sets of hardware. In order to utilize the resources optimally, the test systems might

[1]This definition partially overlaps with definitions from both the International Software Testing Qualifications Board (ISTQB) and the ISO/IEC/IEEE 29119-1 standard [58, 131]. Many other definitions exist.

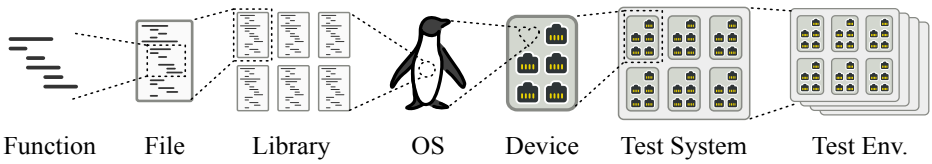


Figure 1.4: Perspectives of testing: from low level (source code functions) to high level (system).

be shared: humans use them by day and machines run nightly testing when no one is in the office. This way developers and testers could use the test systems for development, fault finding and manual testing. During the nights, the organizations would benefit from potentially massive and broad automated testing. This type of testing, on this high level, could be called automated system level testing in the context of networked embedded systems. This perspective is illustrated in the rightmost parts of Figure 1.4.

Wiklund et al. identified lack of time for testing as an important challenge for test automation [135]. One reason could be that system level testing is slow compared to unit testing where a function, file or library may be tested (left part of Figure 1.4). A typical test case for networked embedded systems on system level could be a firewall test. This test case would need at least three devices under test (DUTs): a firewall node, an internal node (to be protected by the firewall), and an external node (trying to reach the internal node). Depending on configuration and type of traffic, the outside node should or should not be able to reach the inside node. These test cases need time because they perform several configuration and verification steps, send actual traffic in the network, and the test framework analyzes the traffic sent. Because testing is slow, it may not be feasible to run all test cases every night, so the organizations need to decide on which test cases to include in or exclude from testing. This problem of regression test selection (RTS, or SL-RTS for System Level RTS) is well-studied, but very little previous research had been done on SL-RTS in 2016 when Paper B was published.

Wiklund et al. also found that low availability of the test environment is an important challenge for test automation [135], in particular if the test systems have non-standard hardware [80]. There is thus a need to maximize the value of the testing, given available resources. If a large number of physical test systems are built, and a large number of test cases designed, then one would need a way to assign the hardware resources of the test systems to the test cases, so that as many test cases as possible could run on as many test systems as possible. In this thesis, this process is referred to as "mapping" of test cases onto test systems. If there is no automated mapping, then each new test case

might trigger a need to build a new test system, which would be expensive and limit the scalability of the testing. Also, in order to maximize the value of the testing, the mapping process should make sure that, over time, each test case utilizes different parts of the hardware in the test systems, such that hardware coverage is increased.

A literature study from 2017 on continuous practices identified that, as the frequency of integrations increase, there is an exponential growth of information [108]. Similarly, Brandtner et al., identified that relevant information might be spread out in several systems [13]. One way to handle test results data is to implement a test results database (TRDB). The TRDB could support exploration, visualization and also provide overviews of results, such that when engineers come to work in the mornings, they could rapidly understand the results of the nightly testing. It is therefore important to study and learn from industrial practitioners how the information flows in their testing processes, and how they make decisions based on visualizations and other support systems enabled by the TRDB.

Intuitively, testing might be expected to be a deterministic process – given the same stimuli to the same system, one could expected the same response. However, this is not true, or rather: in practice, no one is able to observe or control all variables that are of relevance to a complex system, therefore testing is perceived as non-deterministic when some tests intermittently fail. This has been well studied, e.g. by Luo et al. [76] and has been shown to be linked to poor quality test code [42]. However, most of the previous work has been conducted on unit level testing of open source software. It appears as if only four previous studies have considered higher level testing [2, 29, 70, 128].

Chapter 2

Research Process

The research of this thesis has largely followed the process described in Figure 1.1. The common problem has been that of automated system-level software testing of industrial networked embedded systems. The work has been funded by Westermo and the Swedish Knowledge Foundation through the ITS ESS-H research school, and collaborative work has been done, primarily between the Software Testing Laboratory at Mälardalen University and Westermo. Through the research effort new knowledge has been generated, which gives benefits and value to the company, the researchers, and for society in general. The knowledge has led to improvements in work processes and tools used at Westermo, and to academic publications. The value and benefits have led to an interest in continued collaborative research.

The **research goal** of this thesis is: *to improve automated system-level software testing of industrial networked embedded systems.* The five research questions of this thesis are organized to align with Figure 2.1, which shows the overall information flow model discovered in Paper A. In short, this thesis targets five questions that can be positioned in different stages of this flow model: How does information relevant to testing flow in an organization developing embedded systems? Which test cases should be executed when there is not enough time? Which hardware devices should be included in the testing when there are too many possible combinations? Why do some test cases intermittently fail? How can information relevant to testing be presented to developers and testers such that they can make decisions? The rest of this chapter describes these five questions in more detail, the case study research method, as well as a discussion on research quality.

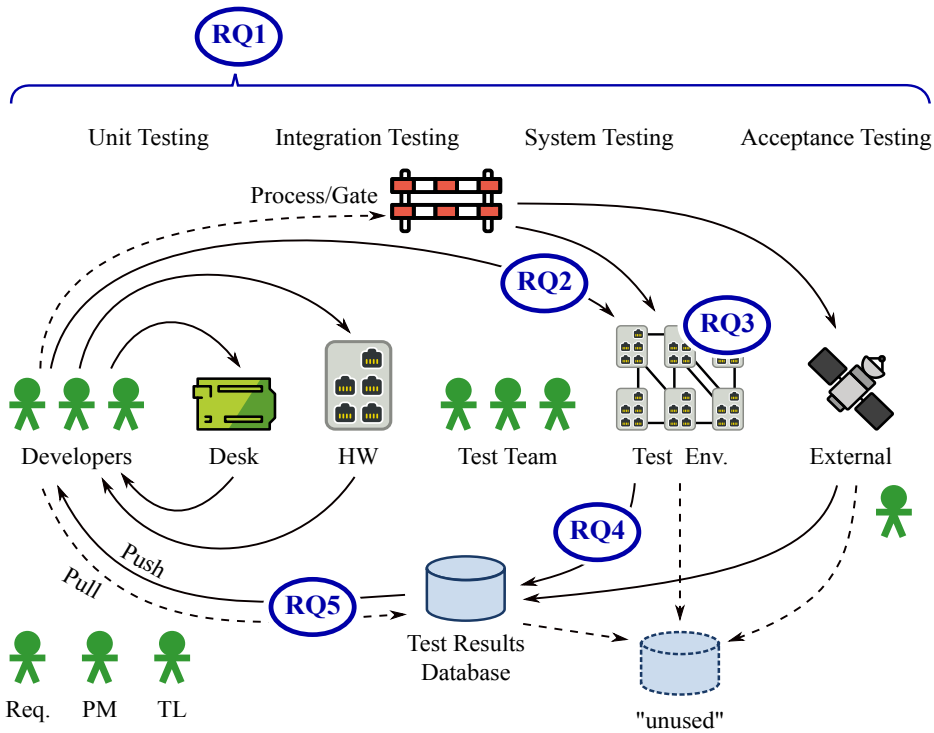


Figure 2.1: Automated System-Level Software Testing of Industrial Networked Embedded Systems with highlighted research questions (compare Figure 1).

2.1 Research Questions

A challenge with testing embedded systems is understanding how updated software reaches a test environment, how testing is actually conducted in the test environment, and how test results reach the developers that updated the software in the first place (i.e. the elements and arrows in Figure 2.1). The following research question (RQ) was formulated in order to tackle this problem:

RQ1: How could one describe the flow of information in software testing in an organization developing embedded systems, and what key aspects, challenges, and good approaches are relevant to this flow? (Paper A)

Despite a long history of research, the regression test selection problem was relatively unexplored from a system-level perspective at the time of Paper B (2016). In Figure 2.1, test selection is illustrated as a process occurring before testing and before software reaches the test environment. The following RQ was formulated in order to explore this problem from a system-level:

RQ2: What challenges might an organization have with respect to system-level regression test selection in the context of networked embedded systems, and how could one address these challenges? (Paper B)

Similar to how regression test case selection occurs prior to the actual testing, a subset of the available hardware must also be selected for each test case – the *mapping* of requirements of the test cases onto test systems. In Figure 2.1, a mapping can be thought of a process requiring a test environment. This leads to the following RQ:

RQ3: What challenges might an organization have with respect to test environment assignment in the context of networked embedded systems, and how could one address these challenges? (Paper C)

When system-level test cases are executed over time, night after night, on embedded systems in an industrial setting, there is almost always a drift in the software, testware or hardware between test sessions. With the exception of mostly manual debugging and trouble shooting, rarely do companies invest effort in doing regression testing of unchanged systems. Instead, one desires confirmation that recent changes have not introduced regressions. Under these conditions, test cases may sometimes intermittently fail for no obvious reason, leading to unexpected test results over time in the test results database (Figure 2.1). This problem motivates the following RQ:

RQ4: What are the root causes of intermittently failing tests during system-level testing in a context under evolution, and could one automate the detection of these test cases? (Paper D)

System-level software testing of industrial networked embedded systems produces large amounts of information, and involves many automated decisions. In the software development process, the results may be hard to render into something actionable, in particular in the daily activities of a software developer, when a new feature is about to be completed, and at release time. This leads to the final RQ of this thesis:

RQ5: How could one implement and evaluate a system to enhance visualization and exploration of test results to support the information flow in an organization? (Paper E)

2.2 The Case Study Research Method

According to Runeson et al. [105], case study research in software engineering is useful in order to understand and improve software engineering processes and products. They define the case study research method as:

> an empirical enquiry that draws on multiple sources of evidence to investigate [...] a contemporary software engineering phenomenon within its real-life context, especially when the boundary between phenomenon and context cannot be clearly specified.

The studies in this thesis are all case studies, but differ in terms of aim and type of data used – both qualitative and quantitative data has been collected and analyzed. Paper A exclusively used qualitative data (in the form of interviews), Papers C and D primarily used quantitative data, and Papers B and E combine qualitative and quantitative data.

When Paper A was being planned, a number of guidelines were considered. Kitchenham and Pfleeger published a series of guideline papers in 2001 and 2002 [65, 66, 67, 68, 95, 96]. These had an influence on, for example, Runeson et al. [105], that in turn has had an influence on the overall guideline paper followed for Paper A: Linåker et al. [72].

A total of about 28 hours of audio and video has been recorded and transcribed into about 185 pages of text for this thesis (17 hours of audio transcribed into 130 pages of text for Paper A, 7 hours of video transcribed into 36 pages of not as dense text for Paper E, and 4 hours of interview audio and 20 pages of text was collected for Paper X3, on which Paper E is based). For the data analysis of this qualitative data, many options were available, e.g. grounded theory (for example described by Stol et al. [114]), content analysis (as described for the field of nursing by Graneheim and Lundman [46]), and thematic analysis. There is a significant overlap between these methods, and also a drift of guidelines into software engineering from the social sciences (e.g., the Cruzes and Dybå guidelines for thematic analysis [22]). In the initial stages of data analysis for Paper A, we held a workshop among the authors where the decision to use the recommendations of thematic analysis described for the field of psychology by Braun and Clarke [14] was taken. Among the authors we saw this approach as easy to understand and suitable for the type of data at hand.

At the time, I (and the other authors too) felt that the integrity and anonymity of the interviewees was of great importance. The interviewees felt the same, and some of them pointed out that leaked transcripts could

uniquely identify them, or be a form industrial espionage. As part of the data collection process of Paper A, an anonymization process based on existing ethical guidelines, e.g. [127], was designed. This, in turn, lead to Paper X5 on ethical interviews in software engineering.

At the core of Paper B is an algorithm and tool for SL-RTS that was evaluated with data from four years of nightly testing – two years of data from before introduction of the tool and two after. Paper C proposes and describes the implementation of an algorithm and tool for hardware selection, which was evaluated using 17 test systems and more than 600 test cases for a total of more than ten thousand possible mappings. This way, performance of the new tool could be compared with the previous tool in use at Westermo. Paper D describes a novel algorithm to identify intermittently or consistently failing tests. The algorithm was used on test results from nine months of nightly testing and more than half a million test verdicts were used in order to identify test cases to investigate further. The extended analysis used project artifacts, such as code commit messages, bug trackers and so on, in order to find patterns for why test cases are intermittently failing. Finally, Paper E investigates how to implement visualizations based on test results, which again revolves around the development of a tool. The tool was evaluated by transcribing recorded reference group meetings, by member checking and by analyzing log files from using the tool.

2.3 On Research Quality

The term *software engineering* was coined by Margaret Hamilton while working on the Apollo program at NASA [54] in an attempt to bring discipline, legitimacy and respect to software. As pointed out by Felderer and Travassos, software engineering has evolved from its seminal moments at NASA (and NATO) aiming at observing and understanding software projects in 1960s and 1970s, via attempts to define methods for empirical studies in the 1980s (such as the work on goal-question-metric by Basili and Weiss [9]), to the formation of publication venues for empirical software engineering in the 1990s, into an evidence-based field of research [35]. In 2012, Runeson et al. published a book called "Case Study Research in Software Engineering – Guidelines and Examples" [105] which is one of the main guidelines used in this thesis.

As mentioned above, Arora et al. [3], argues that one of the strengths of research at companies is the access to data. However, when a company uses its own data and tools to make research claims such as in this thesis, e.g. by using the SuiteBuilder tool "two thirds of the failing tests are now positioned in the first third of the test suites," then this claim cannot easily be

confirmed or rejected by someone not at the company. The term falsifiability was coined by Popper in 1934 [98], and it is sometimes used as a possible demarcation between science and pseudoscience. Without access to company data and tools, would it be any easier to falsify the statement about distribution of failing tests, than it would be to falsify the claim that a teapot orbits the Sun somewhere between Earth and Mars (Russell's teapot)? Intuitively, one might want to argue that this thesis is scientific, but that belief in Russel's teapot is pseudo-scientific. Furthermore, in his famous paper "Why Most Published Research Findings Are False" [57], Ioannidis argues that lack of replication, lack of publications with negative results, and over-emphasis on statistical significance are root causes for why many research findings are false. Again, the statement on the fail distributions is not only hard to falsify, it is also hard to replicate. Claims that are easy to replicate are of course easier to falsify. In retrospect, now in 2021, one would argue that we as authors of the papers in this thesis, should have made a greater effort to make data and tools available to a greater audience. Or that we, in addition to exploring challenges at one or a few companies, would also have used publicly available sources such as open source projects to evaluate findings, tools or hypotheses. If we had, then the opportunities to replicate research findings and the generalizability of the findings would have been better.

It has been suggested that good research should have both rigor and relevance. By doing so, one avoids conducting research only for the sake of researchers. At the core of *rigor* are carefully considered and transparent research methods [60, 105, 106, 132].

Doing research in a certain context certainly has an impact on *generalizability*. However, Briand et al. [16] argue that "we see the need to foster [...] research focused on problems defined in collaboration with industrial partners and driven by concrete needs in specific domains [...]" In other words, research in a narrow context may have high *relevance*, regardless of its generalizability.

Research involving human practitioners may harm these individuals, and *ethical aspects* are of relevance to businesses as well. Smith [110] points out that being an industrial doctoral student comes with unique ethical risks, i.e. an industrial doctoral student might portray his of her employer in a favorable light. For the research in this thesis, we have had a focus on minimizing harm, which has led to guidelines not part of this thesis (in papers X5 and X6 [117, 122]). In addition to avoiding harm to participants in the research, another ethical aspect of this thesis is the potential for hiding decision making criteria of an algorithm, which risks making decisions inscrutable, and thereby disempowering stakeholders [81].

One challenge with the case study research method in software engineering is that when exploring a "phenomenon [...] when the boundary between phenomenon and context cannot be clearly specified" (as Runeson et al. [105] defines a case study), the findings might depend on the context and not the phenomenon. In industrial case studies, many variables may have an impact on what is being measured. There could therefore exist confounding factors, perhaps changes in a software development process, that have not been investigated, or other threats to *internal validity*. However, this threat has been addressed by involving both qualitative and quantitative research methods, and by collecting data as to minimize the impact of confounding variables. E.g., in Paper B, challenges with respect to regression test selection were identified qualitatively. Paper D used a quantitative metric for test case intermittence that led to the identification of test cases that were then qualitatively investigated. Furthermore, prolonged involvement and industrial experience should also have addressed this threat.

In the studies in this thesis, a number of constructs are used (e.g., a flaky test, the mapping problem, and information flow). These do not always have a standard definition, which leads to threats to *construct validity*, not only in the research in this thesis, but in general in the research community. One way to combat this threat to validity is to carefully define terms, and be transparent about algorithms used to collect data and also how qualitative data has been analyzed.

In all but one of the papers of this thesis, the results stem from one context: the automated system-level testing at Westermo. One should therefore consider *external validity* before transferring results into other contexts. However, there is a growing body of knowledge of industrial testing and system-level testing. It is also interesting to speculate about generalizability, e.g. on test selection or intermittently failing tests, from unit-level testing to system-level testing. One such example could be the finding in Paper D that test case assumptions (e.g., on timing) is a factor for intermittence that seems to be common for both unit- and system-level testing.

By being transparent in the choice of, and strict in the use of, scientific methods, while at the same time considering humans and organizations, the research in this thesis should be not only rigorous and relevant, but also ethical.

Chapter 3

Related Work

The research fields of software testing, regression test selection, intermittently failing tests as well as communication and visualization are all well studied. However, there is a notable gap between academia and industry in these fields: an overwhelming majority of the actual testing in the industry is manual, despite more than 40 years of research on regression test selection very little of the research targets the system level, intermittently failing tests have almost exclusively been explored on unit level testing of open source software, and previous work on the role of communication in software engineering and software testing indicate that a requirements specification is of great importance – in practice these are unfortunately typically of poor quality. This chapter discusses findings in these areas of research, and also briefly mention the mathematical field of graph theory and the subgraph isomorphism problem – topics related to Paper C.

3.1 Industrial Software Testing

A lot of research has shown that industry seems to be slow to adopt state of the art techniques for software testing. Most testing is done manually, perhaps as much as 90% [63]. Well-defined test design techniques exist, but testers "rely on their own experience more" [56]. The industrial use of issue trackers[1] is sometimes flawed, and many organizations don't use one [63].

On the positive side, companies change over time and many strive towards using more agile practices, such as shortening feedback loops and adopting continuous integration [89]. In a literature study on continuous practices from

[1] Issue tracking can also be referred to as bug tracking, defect reporting, or incident reporting [59, 87, 140].

2017, Shahin et al. identified that, as code integrations become more frequent, the amount of data such as test results will increase exponentially [108]. Therefore it is "critical to collect and represent the information in [a] timely manner to help stakeholders to gain better and easier understanding and interpretation of the results..." There is also a need for speeding up continuous integration [112].

Agile software development practices can coexist with traditional approaches, and they seem to benefit from each other. Notander et al. points out that a "common belief is that agile processes are in conflict with the requirements of safety standards... Our conclusion is that this might be the case [sometimes], but not [always]" [86]. Ghanbari came to similar conclusions [45]. A study on communication in agile projects suggest that plan-driven approaches are sometimes needed in agile contexts [97]. Similarly, a recent study by Heeager and Nielsen identified four areas of concern when adopting agile practices in a safety critical context (documentation, requirements, life cycle and testing) [51].

When it comes to testing of embedded systems, an important aspect is to investigate non-functional qualities such as timing, and by testing on real hardware one can achieve this [7, 41]. One could also alternate between testing on hardware, virtual hardware and no hardware [21].

In a literature study from 2019, bin Ali et al. involved practitioners in the research process in order to identify industry-relevant research on regression testing [10]. Among other things, they recommend researchers to: evaluate coverage of a technique at the feature level (other coverage metrics were not seen as relevant for practitioners), report on relevant context factors in detail as opposed to reporting generally on many factors and to study people-related factors.

3.1.1 Regression Test Selection

When organizations move towards nightly testing, and run testing on the embedded systems, they risk ending up with nights that are not long enough – too many test cases and too little time. This is a strong motivator for introducing regression test selection (RTS). A well-cited paper by Yoo and Harman proposes three strategies for coping with RTS: *Minimization*, *Selection*, and *Prioritization* [138]. RTS can be based on many different properties: code coverage [83], expected fault locations [91], topic coverage [52], or historic data such as last execution, fault detection, and coverage data [30, 32, 52, 64]. Mathematical optimization approaches have been used [36, 53, 83] as well as genetic algorithms [133]. Software fault prediction is a related field, where

one influential paper was written by Ostrand et al. for traditional software development in large software systems [91]. Elbaum et al. mentions that traditional regression testing techniques that rely on source code instrumentation and availability of a complete test set become too expensive in continuous integration development environments, partly because of the high frequency of code changes [30].

In 2017, when preparing my licentiate thesis, I identified four recent surveys on RTS, none of which was applicable to system-level RTS. The most recent one, from 2016 by Hao et al., mentions that most techniques have been evaluated with programs smaller than 6 kSLOC (SLOC = source lines of code) [49]. This is still very far from many embedded systems where using the Linux kernel alone results in a code base in the order of 10 MSLOC. Second, Catal and Mishra found that the dominating techniques for RTS are coverage based [19] and these can therefore be difficult to apply to system-level software testing due to the instrumentation required. The third survey, by Yoo and Harman, discuss a design-based approach to run what seems to be unit-level test cases: "Assuming that there is traceability between the design and regression test cases, it is possible to perform regression test selection of code-level test cases from the impact analysis of UML design models" [138]. Finally, Engström et al. published a paper in 2010 in which they investigated 28 techniques, where 14 were on statement level, the others were no higher than module level [33].

Now, in 2021, the body of knowledge on system-level RTS has grown and is still growing. Ricken and Dyck did a survey on multi-objective regression test optimization in 2017 [101] and they argue that duration needed for test cases should be taken into account. Lou et al. [75] did a literature study on regression test prioritization. They found that most approaches are not much better than simple ones, the approaches target domains where testing is so fast that selection is not really relevant, and that no free and suitable tool exists (e.g. a tool integrated with jUnit). In a 2020 doctoral thesis on test selection, Haghighatkhah [48] proposes a RTS approach that combines diversity (using test case distances) and historic test data. Similarly, diversity has also been proposed by Neto et al. [84]. In a systematic mapping study from 2020 on test case prioritization in continuous integration environments, Prado Lima and Vergelio found that history-based approaches is a dominating approach in recent publications on RTS, few of the publications they processed report on challenges in testing in a continuous integration context and that future work should also include aspects of intermittently failing tests (flaky tests) as well as time constraints [99].

Paper B builds upon the existing body of knowledge in the field of RTS, as

it was in 2016, and the paper shows feasibility of system-level RTS by using a framework of prioritizers that each investigate some priority-giving aspect. Had the SL-RTS been implemented from scratch today, then one could have also considered the diversity of the selection (see, e.g., Haghighatkhah [48]), and made the tool more scalable as to avoid making a test results database a bottleneck when test complexity increases (see Figure 1.3). The SuiteBuilder tool described in Paper B, has had its database questions simplified a few times in the seven years that have passed since its implementation. A migration from the old database schema and programming language has started, as described in Paper E.

3.1.2 Test Environment Assignment

The body of knowledge with respect to assigning test environments to test cases for execution seems to be in its infancy. While Paper C was in submission, Kaindl et al. published work on the same high level problem. In their paper they "briefly sketch the technical essence of [their] project" explaining that they plan to use a semantic specification, an ontology, and a taxonomy to assign test environments to a test case [62]. However, it seems as if Paper C is the only study with a working solution for this problem in the domain of networked embedded systems.

Paper C investigates and discusses "the mapping problem" and how it is related to the subgraph isomorphism problem. The subgraph isomorphism problem is a well-studied topic in the field of graph theory, and much of the basic terminology is covered in books such as Wilson's introduction to graph theory [136]. There are a number of approaches to solving the subgraph isomorphism problem: satisfiability, for example, is covered by both Knuth and Ullman [69, 129]. Another approach is to do a combinatorial search, such the one described by Bonicci et al. [12]. In the test cases and test systems at Westermo, there are frequently cycles. If there is a cycle in a test case, then this must be mapped onto a cycle in a test system. Algorithms for identifying cycles are therefore of importance for improving the mapping, e.g. work by Paton, or Yuster and Zwick [92, 139].

3.1.3 Intermittently Failing Tests

According to Fowler [37, 38], a *code smell* is a "surface indication" of a potential problem in code. *Smelly tests* represent code smells in test code, such as test cases with poor design or implementation choices. An example of a test smell is "the local hero" test case that will pass in a local environment, but fail

when testing in the target environment because of undocumented dependencies that were not satisfied [42]. A *flaky test* is a test that has been executed at least twice and provided both passing and failing verdicts, without any changes being made to the underlying software, hardware or testware between the two executions [76]. In other words, a flaky test is a test case in which there is an apparent non-determinism in the underlying software, hardware or testware. Flaky tests have been widely studied, and many tests are flaky because they are also smelly [130], and sometimes the removal of test smells leads to removal of flakiness.

Because of the requirement of unchanged software, hardware and testware, the construct of a flaky test is not very convenient in a resource constrained continuous integration environment. Therefore, Paper D defines an *intermittently failing test* as a test case that has been executed repeatedly while there is a potential evolution in software, hardware and/or testware, and where the verdict changes over time.[2] The paper also defines a metric to quantify test cases as more or less intermittently failing based on Markov chains [79, 100]. A similar metric was presented by Breuer in 1973 [15], but discovered independently, and created for another domain – Paper D targets regression testing of software-intense embedded system as opposed to describing faults in hardware. Breuer seems to have been first to use Markov chains to describe faults in a system, as opposed to modeling the system. Another way of quantifying flakiness, based on entropy, has been presented by Gao [40]. This method requires code instrumentation (code coverage) and was evaluated using Java programs with 9 to 90 thousand lines of code. One perceived advantage of Gao's metric is that it can be used to "weed out flaky failures" whereas our approach targeted a better understanding of the root causes of intermittently failing tests.

Lou et al. [76] did an investigation on unit tests in open source projects. They found that many tests were flaky because of asynchronous waiting, concurrency, or test order. Important problems with flaky tests, on this level, are that they can be hard to reproduce, may waste time, may hide other bugs and they can reduce the confidence in testing such that practitioners ignore failing tests [76]. Despite a large amount of research on flaky tests on unit level, only four studies targeting intermittently failing tests on a system level were identified when Paper D was submitted: Ahmad et al. [2], Eck et al. [29], Lam et al. [70], and Thorve et al. [128]. These five studies combined allowed for identification of similarities and differences between factors leading to intermittently failing tests on unit and system levels.

[2] A recent paper by Barboni et al. [8] explores different definitions and overlapping aspects of the terminology of flaky or intermittently failing tests.

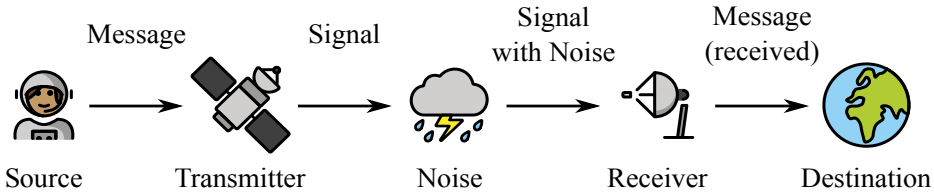


Figure 3.1: The Shannon-Weaver communication model.

3.2 Communication and Flow of Information

In a 60 year old publication [109] Shannon first described what would later be known as the Shannon-Weaver model of communication, shown in Figure 3.1. This model has later been built upon to represent countless variants. It contains an information source converted into a message that a transmitter converts to a signal, noise is added to the signal from a noise source. The signal and message then reach the destination. Later research on this model add, among other things, medium used, as well as the role of distances.

Information flow has an important overlap with the concept of communication, it can be defined by a distributed system of agents and the relationships between them. Information flow is important when a synergy between humans and software systems is required for a work flow [26].

A theory for *distances* in software engineering "explains how practices improve the communication within a project by impacting distances between people, activities and artifacts" [11]. As an example, the practice of *Cross-Role Collaboration* has an impact on the temporal distance. Olson and Olson found that new technology may make geographical distance smaller, making cultural distances appear to be greater [88]. They also found that modern communication media frequently relies on technical solutions, such as a conference phone, and when there are usability problems, the users adapted their behavior instead of fixing the technology – by shouting in the conference phone.

The requirements specification is the most important document for testers during system testing, but this document is often of poor quality [56], and is one of the main sources of technical debt for safety development [45]. Alternatively, documentation can be seen as a complement to communication that can help combat knowledge evaporation [77]. Human factors and the quality of the requirements specifications were also mentioned in a qualitative interview study "as essential for development of safety critical systems" [86]. Furthermore, the Annex E of the first part of the ISO/IEC/IEEE 29119 standard acknowledges that testers need to communicate, with various stakeholders, in a timely manner, and that this communication may be more formal (e.g., written

reports) or oral (as in agile contexts) [58]. Similarly, the International Software Qualifications Board, ISTQB, released their syllabus for certification of test automation engineers [6] in 2016. It covers topics that frequently occurred in our interviews in Paper A. They recommend that one should: measure benefits of automation (costs are easily seen), visualize results, create and store logs from both the system under test and the test framework, and generate reports after test sessions.

Papers A and E extend the body of knowledge in the field of communication and information flow in software engineering with results on a holistic approach to the flow of information in software testing, as well as a tool for test results exploration and visualization.

3.3 Visualization

According to Diehl, there are three important aspects of software visualization: structure, behavior and evolution [24]. An early visualization technique to show structure is from 1958 and generated control-flow diagrams because it "is not practical to expect [everyone] to be intimately familiar with [all code]" [107]. Visualizations may not only show structure but also compare structures. One survey on software visualization, present methods for visualizing static aspects with the focus of source code lines, class metrics, relationships, and architectural metrics [18].

In a well-cited paper from 2002, Jones et al. use test results data with a combination of source code, code coverage, unit tests and test results in a sort of an extended Integrated Development Environment (IDE) view [61]. A dashboard is a common approach for quality monitoring and control in the industry. These aim at presenting one or more key performance indicators (KPIs) over time [23, 39]. When visualizing test results, there is a need for summaries [85], information may be spread out in several different systems [13], one might need to consider transitions between different views of a test results visualization tool [90], and one should consider that visualizations may target different stakeholders for different purposes [115].

Rosling et al. argues that "The world cannot be understood without numbers. But the world cannot be understood with numbers alone." The Roslings developed software to visualize the improvements in world health because students were found to perform worse than random in quizzes on the topic [103, 104]. This could be seen as an indication of the importance of visualizations for comprehending data. In his keynote at the practitioner-oriented conference Beauty in Code 2018 [5], James Bach highlighted that stakeholders do not know what to visualize: "What I had to do, is use my skill as a tester,

and my interest in visual design, and complexity, and displays, and statistics, to try to come up with something, that they, when they saw it – it would be like someone seeing an iPhone for the first time, they would say 'I used to like Nokias and now I like Apple phones'." This could be seen as supporting findings in Paper A, that visualizations are not prioritized and an individual skilled in other domains ends up being the one preparing visualizations.

Adopting a data science approach at a company can be hard – data science deals with collection, processing, preparation and analysis of data, it is a cross-domain discipline that requires skills beyond software engineering and computer science, e.g. mathematics and domain knowledge [28]. Paper E strives to make test results data visualized and explorable. Similar goals were held in the q-rapids project[3] where data was collected, processed and then shown to product owners with the aim of enhancing their cognition, situational awareness, and decision-making capabilities [20]. Some of the challenges they have seen in their work on q-rapids are: to be transparent about what a value or metric means and its origins, integration with the tool and other tools, as well as the need for the tool to be tailored to a company. Some of the lessons learned were to use a common entry point for the data, and that implementation of the tool requires experts (e.g. when setting up data collection streams and doing analysis of data) [82]. As opposed to the q-rapids project, the tool described in Paper E does not strive to distill test results data into key performance indicators. Instead, it targets exploration and visualization and, in some sense, interaction with the data. Ahmad et al. [1] investigated information needs for testing and identified eight needs that closely match the work in Paper E, such as being aware of if, and if so where (on which test system and code branch), a test case fails.

With respect to the body of knowledge in the field of visualization of software test results, Paper E confirms some of the previous work (such as the importance of visualizations, and the need to consider transitions between views). It also provides a long term case study of the implementation and usage of test results visualization and decision making, and shows the importance of the TRDB.

[3] Q-Rapids: Quality-aware rapid software development: www.q-rapids.eu.

Chapter 4

Contributions

The research goal of this thesis is *to improve automated system-level software testing of industrial networked embedded systems.* To reach the goal, the thesis poses five RQs that have been targeted with five studies. Each paper primarily targets one RQ (RQ1: Paper A, RQ2: Paper B, etc.). This chapter revisits the RQs and summarizes their main contributions (C).

4.1 RQ1: Information Flow

How could one describe the flow of information in software testing in an organization developing embedded systems, and what key aspects, challenges, and good approaches are relevant to this flow?

Paper A seems to be the first study to take a high level approach to the flow of information in software testing. For the study, twelve interviews with industry practitioners were held. The interviewees were from five different companies and had an average of more than 14 years of experience. 17 hours of audio was recorded, anonymized and transcribed into 130 pages of text, that was analyzed with thematic analysis. The main contributions of Paper A are:

C-A1: An overall model of the flow of information in software testing.

C-A2: Six key factors that affect the flow of information in software testing (how organizations conduct testing and trouble shooting, communication, processes, technology, artifacts, as well as how it is organized).

C-A3: Seven main challenges for the information flow (comprehending the objectives and details of testing, root cause identification, poor feedback, postponed testing, poor artifacts and traceability, poor tools and test infrastructure, and distances).

C-A4: Five good approaches for enhancing the flow (close collaboration between roles, fast feedback, custom test report automation, test results visualization, and the use of suitable tools and frameworks).

4.2 RQ2: Test Selection

What challenges might an organization have with respect to system-level regression test selection in the context of networked embedded systems, and how could one address these challenges?

Paper B seems to be one of the first papers on system-level regression test selection. The study covers three industrial challenges with nightly regression testing: nightly testing not finishing on time, manual work and omitted tests, as well as no priority for the test cases. These problems were solved by implementing SuiteBuilder. The algorithm was evaluated quantitatively using data from four years of nightly testing, as well as qualitatively with interview data. The main contributions of Paper B are:

C-B1: The SuiteBuilder tool, a framework of prioritizers assigning priorities based on multiple factors.

C-B2: Empirical evidence that the tool improves the testing process: test suites finish on time, and that two thirds of the failing tests are now positioned in the first third of the test suites.

4.3 RQ3: Hardware Selection

What challenges might an organization have with respect to test environment assignment in the context of networked embedded systems, and how could one address these challenges?

When performing testing of embedded systems, it is critical to also involve real hardware, and Paper C shows what appears to be the first evaluated approach to solve the mapping problem. The process of mapping involves a way to map a test case onto a test system using graph theory, in particular the graph models of the test systems and test cases. This way the subgraph isomorphism problem is adapted to the context of networked embedded devices. The prototype implementation was evaluated quantitatively with the available test systems and test cases for more than 10.000 different pairs of graphs. The main contributions of Paper C are:

C-C1: A new mapper, that was found to be more than 80 times faster than the old tool.

C-C2: An extension of the mapper, where the TRDB with previous mappings is used to map in *different* ways over time such that the DUT coverage grew from a median of 33%, to a median of 100% in just five iterations.

4.4 RQ4: Intermittently Failing Tests

What are the root causes of intermittently failing tests during system-level testing in a context under evolution, and could one automate the detection of these test cases?

Paper D defines a novel metric for intermittently failing tests. By analyzing more than half a million test verdicts from nine months of nightly testing, both intermittently and consistently failing tests were identified. The main contributions of Paper D are:

C-D1: A novel metric that can identify intermittently failing tests, and measure the level of intermittence over the test base.

C-D2: Nine factors that may lead to intermittently failing tests for system-level testing of an embedded system (test case assumptions, complexity of testing, software and/or hardware faults, test case dependencies, resource leaks, network issues, random numbers issues, test system issues, and refactoring).

C-D3: Evidence that finding the root cause of intermittently failing tests often involves more effort, when compared to root cause analysis of consistently failing tests.

C-D4: Evidence that a fix for a consistently failing test often repairs a larger number of failures found by other test cases than a fix of an intermittently failing test.

4.5 RQ5: Test Results Exploration and Visualization

How could one implement and evaluate a system to enhance visualization and exploration of test results to support the information flow in an organization?

Paper E focuses on a new tool, Tim, that was implemented to replace an old system for test results exploration and visualization (TREV). Work was prioritized and evaluated with a reference group of 12 individuals. From reference group meetings, 7 hours of video was recorded and transcribed into 36 pages of text, and logs from 201 days of using Tim was also collected. The main contributions of Paper E are:

C-E1: Four solution patterns for TREV (filtering, aggregation, previews and comparisons).

C-E2: Implementation and empirical evaluation of eight views for TREV (start, outcomes, outcome, session, heatmap, measurements, compare branch and analyze branch views).

C-E3: Six challenges for TREV (expectations, anomalies, navigation, integrations, hardware details and plots).

Chapter 5

Future Work

This thesis covers system-level test automation of embedded systems, and in particular test results information flow, regression test selection, hardware selection, intermittently failing tests as well as test results exploration and visualization. This chapter discusses some of the potential for future research.

The topics of test results information flow and test results exploration and visualization are partly overlapping. Future work could investigate, on a higher level, how an organization would benefit from automated test reporting, as well as if or how humans and such a system could co-create meaningful test reports – would this be a way to bridge some of the challenges of combining agile and traditional development?

Despite already being very well-researched, there is room for more work on the regression test selection problem – in particular with respect to diversity, what the characteristics of a good selection is, and how different selection strategies impact these characteristics. Perhaps industry and academia could co-create an open source tool that could be adjusted to fit different needs (such as when one organization requires a certain code coverage, whereas another focuses on diversity)? Also, if the test selection strategy was reactive (perhaps search-based) and given the mandate to re-prioritize on the fly during a test session, would that improve test coverage, or other desirables of the testing?

As mentioned above, research on automated hardware selection for testing embedded systems seems to be in its infancy. However, perhaps a more generalized problem of hardware selection could be well served from findings in combinatorial testing of product lines or search-based software engineering [25, 47, 50, 74]? With respect to the results in this thesis, questions that remain unanswered range from fundamental (is test coverage of two different hardware platforms exactly twice as valuable as coverage of one?), via detailed (how different would a tool using satisfiability to solve the mapping problem

be when compared to one that uses the subgraph isomorphism problem, and are they equally explicable to practitioners?), to extremely detailed (how much does the performance of the mapping tool increase if cycles in the test systems and test cases are considered?). The benefits and drawbacks of testing on actual devices with hardware, on virtualized systems, or using emulated hardware would also be interesting to explore. When is it good enough to test without real hardware, and when must it be used?

Similar to the regression test selection problem, the topic of intermittently failing tests is receiving a great amount of research attention. Most of this research is done for unit-level testing, perhaps because most test automation is done at this level and public data sets are available. Some of this research seems to imply that the root cause of the intermittence is in the test cases themselves, which Paper D identified as only one type of root cause. A question that seems to remain unanswered is: which of the root causes to intermittence are generalizable across testing levels and domains?

Rule-based systems, such as the SuiteBuilder tool, is a type of artificial intelligence (AI). More advanced AI learns from data to become increasingly better at solving tasks, sometimes at the price of transparency or explicability [71, 122]. It is very clear that society in general, and software test automation in particular, could gain tremendously by adopting more AI. As this thesis is being written, Westermo has joined an industry-academia research project involving AI (described in [34]). What the exact outcomes will be remains to be seen, but in general, it seems as if almost all aspects of this thesis could benefit from further automation, with or without AI.

Chapter 6

Conclusions

As mentioned in the abstract of this thesis, embedded systems appear almost everywhere and play a crucial role for industrial and transport applications – software testing is crucial for the quality assurance of these systems. This thesis addresses five challenges for automated system-level software testing of industrial networked embedded systems. These challenges have been tackled in five studies, each self-contained with its own conclusion section. To conclude the thesis, this chapter takes a step back, and looks at the big picture. A central result of this thesis is that software testing can be seen as a set of feedback loops, that can be visualized in the flow diagram in Figure 1, which was one of the main findings of Paper A. This study focused on the overall flow of information in software testing. The remaining studies fit well into this picture. Both test selection (Paper B) and test environment assignment (Paper C) can be seen as processes having an impact on *what* to test, and *with what* to test, in the test environment. The role of the test results database has, in one way, an importance *after* the testing (e.g. to aid in reporting), but because it may impact coming test selection and the coming test environment assignment, it also has an importance *before* the testing activity. The impact of intermittently failing tests (Paper D) can be seen as having an influence on test results, how it reaches developers and the distrust of testing it might lead to for developers. The findings on intermittently failing tests also point to a need to improve software, hardware and testware in order to avoid intermittent faults in these. The work on visualizing test results and making them explorable (Paper E) is one approach of improving the feedback loop in Figure 1.

The research goal of this thesis is: *to improve automated system-level software testing of industrial networked embedded systems.* In short, this has been achieved with an improved understanding of the flow of information in software testing, selection of test cases and hardware for testing, with an investi-

gation of intermittently failing tests, as well as improved visualizations of test results.

Bibliography

[1] A. Ahmad, O. Leifler, and K. Sandahl. Data visualisation in continuous integration and delivery: Information needs, challenges, and recommendations. *Wiley IET Software*, 2021.

[2] A. Ahmad, O. Leifler, and K. Sandahl. Empirical analysis of practitioners' perceptions of test flakiness factors. *Wiley Journal of Software Testing, Verification and Reliability*, page e1791, 2021.

[3] A. Arora, S. Belenzon, A. Patacconi, and J. Suh. The Changing Structure of American Innovation: Some Cautionary Remarks for Economic Growth. *University of Chicago Press: Innovation Policy and the Economy*, 20(1):39–93, 2020.

[4] L. Assbring and C. Nuur. What's in it for industry? A case study on collaborative doctoral education in sweden. *Sage Publications: Industry and Higher Education*, 31(3):184–194, 2017.

[5] J. Bach. Beauty or Bugs: Using the Blink Oracle in Testing. Keynote at the Beauty in Code Conference, Malmö, Sweden, 2018.

[6] B. Bakker, G. Bath, A. Born, M. Fewster, J. Haukinen, J. McKay, A. Pollner, R. Popescu, and I. Schieferdecker. Certified Tester Advanced Level Syllabus Test Automation Engineer. Technical report, International Software Testing Qualifications Board (ISTQB), 2016.

[7] A. Banerjee, S. Chattopadhyay, and A. Roychoudhury. On Testing Embedded Software. *Elsevier Advances in Computers*, 101:121–153, 2016.

[8] M. Barboni, A. Bertolino, and G. De Angelis. What We Talk About When We Talk About Software Test Flakiness. In *International Conference on the Quality of Information and Communications Technology*, pages 29–39. Springer, 2021.

[9] V. R. Basili and D. M. Weiss. A Methodology for Collecting Valid Software Engineering Data. *IEEE Transactions on Software Engineering*, (6):728–738, 1984.

[10] 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. *Springer Empirical Software Engineering*, 24(4):2020–2055, 2019.

[11] E. Bjarnason and H. Sharp. The role of distances in requirements communication: a case study. *Springer Requirements Engineering*, pages 1–26, 2015.

[12] V. Bonnici, R. Giugno, A. Pulvirenti, D. Shasha, and A. Ferro. A subgraph isomorphism algorithm and its application to biochemical data. *BioMed Central: BMC bioinformatics*, 14(7):S13, 2013.

[13] M. Brandtner, E. Giger, and H. Gall. Supporting Continuous Integration by Mashing-Up Software Quality Information. In *Software Evolution Week, Conference on Software Maintenance, Reengineering, and Reverse Engineering*. IEEE, 2014.

[14] V. Braun and V. Clarke. Using thematic analysis in psychology. *Taylor & Francis: Qualitative research in psychology*, 3(2):77–101, 2006.

[15] M. A. Breuer. Testing for Intermittent Faults in Digital Circuits. *IEEE Transactions on Computers*, 100(3):241–246, 1973.

[16] L. Briand, D. Bianculli, S. Nejati, F. Pastore, and M. Sabetzadeh. The Case for Context-Driven Software Engineering Research: Generalizability Is Overrated. *IEEE Software*, 34(5):72–75, 2017.

[17] J. C. Carver, O. Dieste, N. A. Kraft, D. Lo, and T. Zimmermann. How Practitioners Perceive the Relevance of ESEM Research. In *International Symposium on Empirical Software Engineering and Measurement*. ACM-IEEE, 2016.

[18] P. Caserta and O. Zendra. Visualization of the Static Aspects of Software: A Survey. *IEEE Transactions on Visualization and Computer Graphics*, 17(7):913–933, 2011.

[19] C. Catal and D. Mishra. Test case prioritization: a systematic mapping study. *Springer Software Quality Journal*, 21(3):445–478, 2013.

[20] M. Choraś, R. Kozik, D. Puchalski, and R. Renk. Increasing product owners' cognition and decision-making capabilities by data analysis approach. *Springer Cognition, Technology & Work*, 21(2):191–200, 2019.

[21] P. Cordemans, S. Van Landschoot, J. Boydens, and E. Steegmans. Test-Driven Development as a Reliable Embedded Software Engineering Practice. In *Embedded and Real Time System Development: A Software Engineering Perspective*, pages 91–130. Springer, 2014.

[22] D. S. Cruzes and T. Dybå. Recommended Steps for Thematic Synthesis in Software Engineering. In *International Symposium on Empirical Software Engineering and Measurement*, pages 275–284. IEEE, 2011.

[23] F. Deissenboeck, E. Juergens, B. Hummel, S. Wagner, B. M. y Parareda, and M. Pizka. Tool Support for Continuous Quality Control. *IEEE Software*, 25(5):60–67, 2008.

[24] S. Diehl. Past, Present, and Future of and in Software Visualization. In *International Conference on Computer Vision, Imaging and Computer Graphics*. Springer, 2014.

[25] I. do Carmo Machado, J. D. McGregor, Y. C. Cavalcanti, and E. S. De Almeida. On strategies for testing software product lines: A systematic literature review. *Elsevier Information and Software Technology*, 56(10):1183–1199, 2014.

[26] C. Durugbo, A. Tiwari, and J. R. Alcock. Modelling information flow for organisations: A review of approaches and future challenges. *Elsevier International Journal of Information Management*, 33(3):597–610, 2013.

[27] T. Dybå, D. I. Sjøberg, and D. S. Cruzes. What Works for Whom, Where, When, and Why? On the Role of Context in Empirical Software Engineering. In *International Symposium on Empirical Software Engineering and Measurement*. ACM-IEEE, 2012.

[28] C. Ebert, J. Heidrich, S. Martínez-Fernández, and A. Trendowicz. Data Science: Technologies for Better Software. *IEEE Software*, 36(6):66–72, 2019.

[29] M. Eck, F. Palomba, M. Castelluccio, and A. Bacchelli. Understanding Flaky Tests: The Developer's Perspective. In *Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2019.

[30] S. Elbaum, G. Rothermel, and J. Penix. Techniques for Improving Regression Testing in Continuous Integration Development Environments.

In *International Symposium on Foundations of Software Engineering*, pages 235–245. ACM, 2014.

[31] S. Eldh. Some Researcher Considerations When Conducting Empirical Studies in Industry. In *International Workshop on Conducting Empirical Studies in Industry*, pages 69–70. IEEE, 2013.

[32] E. Engström, P. Runeson, and A. Ljung. Improving Regression Testing Transparency and Efficiency with History-Based Prioritization – An Industrial Case Study. In *International Conference on Software Testing, Verification and Validation*, pages 367–376. IEEE, 2011.

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

[34] R. Eramo, V. Muttillo, L. Berardinelli, H. Bruneliere, A. Gomez, A. Bagnato, A. Sadovykh, and A. Cicchetti. AIDOaRt: AI-augmented Automation for DevOps, a Model-based Framework for Continuous Development in Cyber-Physical Systems. In *Euromicro Conferenece on Digital Systems Design*, 2021.

[35] M. Felderer and G. H. Travassos. The Evolution of Empirical Methods in Software Engineering. In *Contemporary Empirical Methods in Software Engineering*, pages 1–24. Springer, 2020.

[36] K. F. Fischer. A Test Case Selection Method for the Validation of Software Maintenance Modifications. In *Computer Software and Applications Conference*, volume 77, pages 421–426. IEEE, 1977.

[37] M. Fowler. Codesmell (blog post). `https://martinfowler.com/bliki/CodeSmell.html`, 2006. Online, Accessed 2021-09-08.

[38] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 2018.

[39] M.-E. Froese and M. Tory. Lessons Learned from Designing Visualization Dashboards. *IEEE Computer Graphics and Applications*, 36(2):83–89, 2016.

[40] Z. Gao. *Quantifying Flakiness and Minimizing its Effects on Software Testing*. PhD thesis, University of Maryland, 2017.

[41] V. Garousi, M. Felderer, Ç. M. Karapıçak, and U. Yılmaz. What We Know about Testing Embedded Software. *IEEE Software*, 2017.

[42] V. Garousi and B. Küçük. Smells in software test code: A survey of knowledge in industry and academia. *Elsevier Journal of Systems and Software*, 138:52–81, 2018.

[43] V. Garousi, K. Petersen, and B. Ozkan. Challenges and best practices in industry-academia collaborations in software engineering: A systematic literature review. *Elsevier Information and Software Technology*, 79:106–127, 2016.

[44] L. Geschwind. Doctoral Training in Sweden: History, Trends and Developments. In *Doctoral Education for the Knowledge Society*, pages 35–49. Springer, 2018.

[45] H. Ghanbari. Seeking Technical Debt in Critical Software Development Projects: An Exploratory Field Study. In *Hawaii International Conference on System Sciences*, pages 5407–5416. IEEE, 2016.

[46] U. H. Graneheim and B. Lundman. Qualitative content analysis in nursing research: concepts, procedures and measures to achieve trustworthiness. *Elsevier Nurse Education Today*, 24(2):105–112, 2004.

[47] J. D. Hagar, T. L. Wissink, D. R. Kuhn, and R. N. Kacker. Introducing Combinatorial Testing in a Large Organization. *IEEE Computer*, 48(4):64–72, 2015.

[48] A. Haghighatkhah. *Test case prioritization using build history and test distances: An approach for improving automotive regression testing in continuous integration environments*. PhD thesis, University of Oulu, 2020.

[49] D. Hao, L. Zhang, and H. Mei. Test-case prioritization: achievements and challenges. *Springer Frontiers of Computer Science*, 10(5):769–777, 2016.

[50] M. Harman and B. F. Jones. Search-based software engineering. *Elsevier Information and Software Technology*, 43(14):833–839, 2001.

[51] L. T. Heeager and P. A. Nielsen. Meshing agile and plan-driven development in safety-critical software: a case study. *Springer Empirical Software Engineering*, pages 1–28, 2020.

[52] H. Hemmati, Z. Fang, and M. V. Mantylä. Prioritizing Manual Test Cases in Traditional and Rapid Release Environments. In *International Conference on Software Testing, Verification and Validation*, pages 1–10. IEEE, 2015.

[53] K. Herzig, M. Greiler, J. Czerwonka, and B. Murphy. The Art of Testing Less without Sacrificing Quality. In *International Conference on Software Engineering*, pages 483–493. IEEE Press, 2015.

[54] R. Hind. The First computer on the Moon. *Oxford University Press: ITNOW*, 61(3):22–23, 2019.

[55] S. E. Hove and B. Anda. Experiences from Conducting Semi-Structured Interviews in Empirical Software Engineering Research. In *International Software Metrics Symposium*, pages 10–pp. IEEE, 2005.

[56] T. Illes-Seifert and B. Paech. On the Role of Communication, Documentation and Experience during System Testing – An Interview Study. In *Prozessinnovation mit Unternehmenssoftware Workshop at the Multikonferenz Wirtschaftsinformatik*. Springer, 2008.

[57] J. P. A. Ioannidis. Why Most Published Research Findings Are False. *PLoS Medicine*, 2(8):e124, 2005.

[58] ISO/IEC/IEEE. Software and systems engineering – Software testing – Part 1: Concepts and definitions. ISO/IEC/IEEE Standard 29119-1:2013, International Organization for Standardization, International Electrotechnical Commission, Institute of Electrical and Electronics Engineers, 2013.

[59] ISO/IEC/IEEE. Software and systems engineering – Software testing – Part 3: Test documentation. ISO/IEC/IEEE Standard 29119-3:2013, International Organization for Standardization, International Electrotechnical Commission, Institute of Electrical and Electronics Engineers, 2013.

[60] M. Ivarsson and T. Gorschek. A method for evaluating rigor and industrial relevance of technology evaluations. *Springer Empirical Software Engineering*, 16(3):365–395, 2011.

[61] J. A. Jones, M. J. Harrold, and J. Stasko. Visualization of Test Information to Assist Fault Localization. In *International Conference on Software Engineering*, pages 467–477. ACM, 2002.

[62] H. Kaindl, F. Lukasch, M. Heigl, S. Kavaldjian, C. Luckeneder, and S. Rausch. Verification of Cyber-Physical Automotive Systems-of-Systems: Test Environment Assignment. In *International Conference on Software Testing, Verification and Validation Workshops*. IEEE, 2018.

[63] J. Kasurinen, O. Taipale, and K. Smolander. Software Test Automation in Practice: Empirical Observations. *Hindawi Advances in Software Engineering*, 2010.

[64] J.-M. Kim and A. Porter. A History-Based Test Prioritization Technique for Regression Testing in Resource Constrained Environments. In *International Conference on Software Engineering*, pages 119–129. ACM, 2002.

[65] B. A. Kitchenham and S. L. Pfleeger. Principles of Survey Research Part 3: Constructing a Survey Instrument. *ACM SIGSOFT Software Engineering Notes*, 27(2):20–24, 2002.

[66] B. A. Kitchenham and S. L. Pfleeger. Principles of Survey Research Part 4: Questionnaire Evaluation. *ACM SIGSOFT Software Engineering Notes*, 27(3):20–23, 2002.

[67] B. A. Kitchenham and S. L. Pfleeger. Principles of Survey Research Part 5: Populations and Samples. *ACM SIGSOFT Software Engineering Notes*, 27(5):17–20, 2002.

[68] B. A. Kitchenham and S. L. Pfleeger. Principles of Survey Research Part 6: Data Analysis. *ACM SIGSOFT Software Engineering Notes*, 28(2):24–27, 2003.

[69] D. Knuth. Fascicle 6: Satisfiability, volume 19 of The Art of Computer Programming. Addison-Wesley, Reading, Mass, 2015.

[70] W. Lam, P. Godefroid, S. Nath, A. Santhiar, and S. Thummalapenta. Root Causing Flaky Tests in a Large-Scale Industrial Setting. In *International Symposium on Software Testing and Analysis*. ACM, 2019.

[71] S. Legg and M. Hutter. A collection of definitions of intelligence. *IOS Press: Frontiers in Artificial Intelligence and applications*, 157:17, 2007.

[72] J. Linåker, S. M. Sulaman, M. Höst, and R. M. de Mello. Guidelines for Conducting Surveys in Software Engineering v. 1.1. Technical report, Lund University, Sweden, 2015.

[73] D. Lo, N. Nagappan, and T. Zimmermann. How Practitioners Perceive the Relevance of Software Engineering Research. In *Joint Meeting on Foundations of Software Engineering*, pages 415–425. ACM.

[74] R. E. Lopez-Herrejon, L. Linsbauer, and A. Egyed. A systematic mapping study of search-based software engineering for software product lines. *Elsevier Information and Software Technology*, 61:33–51, 2015.

[75] Y. Lou, J. Chen, L. Zhang, and D. Hao. A Survey on Regression Test-Case Prioritization. In *Advances in Computers*, volume 113, pages 1–46. Elsevier, 2019.

[76] Q. Luo, F. Hariri, L. Eloussi, and D. Marinov. An Empirical Analysis of Flaky Tests. In *International Symposium on Foundations of Software Engineering*, pages 643–653. ACM, 2014.

[77] O. Manai. How software documentation helps communication in development teams: A case study on architecture and design documents. Bachelor's Thesis, University of Gothenburg, 2019.

[78] D. Marijan and A. Gotlieb. Industry-Academia research collaboration in software engineering: The Certus model. *Elsevier Information and Software Technology*, 132:106473, 2021.

[79] A. A. Markov. Extension of the law of large numbers to dependent quantities. *Izv. Fiz.-Matem. Obsch. Kazan Univ.(2nd Ser)*, 15:135–156, 1906.

[80] T. Mårtensson, D. Ståhl, and J. Bosch. Continuous Integration Applied to Software-Intensive Embedded Systems – Problems and Experiences. In *International Conference on Product-Focused Software Process Improvement*, pages 448–457. Springer, 2016.

[81] K. Martin. Ethical Implications and Accountability of Algorithms. *Springer Journal of Business Ethics*, 160(4):835–850, 2019.

[82] S. Martínez-Fernández, A. M. Vollmer, A. Jedlitschka, X. Franch, L. López, P. Ram, P. Rodríguez, S. Aaramaa, A. Bagnato, M. Choraś, and J. Partanen. Continuously Assessing and Improving Software Quality With Software Analytics Tools: A Case Study. *IEEE Access*, 7:68219–68239, 2019.

[83] D. Mondal, H. Hemmati, and S. Durocher. Exploring Test Suite Diversification and Code Coverage in Multi-Objective Test Case Selection.

In *International Conference on Software Testing, Verification and Validation*, pages 1–10. IEEE, 2015.

[84] F. G. D. O. Neto, R. Feldt, L. Erlenhov, and J. B. D. S. Nunes. Visualizing test diversity to support test optimisation. In *Asia-Pacific Software Engineering Conference*, pages 149–158. IEEE, 2018.

[85] A. Nilsson, J. Bosch, and C. Berger. Visualizing Testing Activities to Support Continuous Integration: A Multiple Case Study. In *International Conference on Agile Software Development*, pages 171–186. Springer, 2014.

[86] J. P. Notander, M. Höst, and P. Runeson. Challenges in Flexible Safety-Critical Software Development – An Industrial Qualitative Survey. In *International Conference on Product Focused Software Process Improvement*, pages 283–297. Springer, 2013.

[87] K. Olsen, T. Parveen, R. Black, D. Friedenberg, M. Hamburg, J. McKay, M. Posthuma, H. Schaefer, R. Smilgin, M. Smith, S. Toms, S. Ulrich, M. Walsh, E. Zakaria, T. Müller, A. Beer, M. Klonk, R. Verma, D. Graham, E. van Veenendaal, S. Eldh, M. Pyhäjärvi, and G. Thompson. Certified tester foundation level syllabus. Technical report, International Software Testing Qualifications Board (ISTQB), 2018.

[88] G. M. Olson and J. S. Olson. Distance matters. *Taylor & Francis Human-Computer Interaction*, 15(2):139–178, 2000.

[89] H. H. Olsson, H. Alahyari, and J. Bosch. Climbing the "Stairway to Heaven" – A Multiple-Case Study Exploring Barriers in the Transition from Agile Development Towards Continuous Deployment of Software. In *Conference on Software Engineering and Advanced Applications*, pages 392–399. IEEE, 2012.

[90] R. Opmanis, P. Kikusts, and M. Opmanis. Visualization of Large-Scale Application Testing Results. *University of Latvia: Baltic Journal of Modern Computing*, 4(1):34, 2016.

[91] T. J. Ostrand, E. J. Weyuker, and R. M. Bell. Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering*, 31(4):340–355, 2005.

[92] K. Paton. An Algorithm for Finding a Fundamental Set of Cycles of a Graph. *Communications of the ACM*, 12(9):514–518, 1969.

[93] M. Q. Patton. *Qualitative Research & Evaluation Methods: Integrating Theory and Practice*. Sage publications, 2014.

[94] K. Petersen and C. Wohlin. Context in Industrial Software Engineering Research. In *International Symposium on Empirical Software Engineering and Measurement*, pages 401–404. IEEE, 2009.

[95] S. L. Pfleeger and B. A. Kitchenham. Principles of Survey Research Part 1: Turning Lemons into Lemonade. *ACM SIGSOFT Software Engineering Notes*, 26(6):16–18, 2001.

[96] S. L. Pfleeger and B. A. Kitchenham. Principles of Survey Research Part 2: Designing a Survey. *ACM SIGSOFT Software Engineering Notes*, 27(1):18–20, 2002.

[97] M. Pikkarainen, J. Haikara, O. Salo, P. Abrahamsson, and J. Still. The impact of agile practices on communication in software development. *Springer Empirical Software Engineering*, 13(3):303–337, 2008.

[98] K. Popper. *The logic of scientific discovery*. Routledge, 2005.

[99] J. A. Prado Lima and S. R. Vergilio. Test Case Prioritization in Continuous Integration environments: A systematic mapping study. *Elsevier Information and Software Technology*, page 106268, 2020.

[100] N. Privault. *Understanding Markov Chains: Examples and Applications*. Springer, 2013.

[101] K. Ricken and A. Dyck. A Survey on Multi-objective Regression Test Optimization. *Full-scale Software Engineering/The Art of Software Testing*, pages 32–37, 2017.

[102] P. Rosenkranz, M. Wählisch, E. Baccelli, and L. Ortmann. A Distributed Test System Architecture for Open-source IoT Software. In *Workshop on IoT challenges in Mobile and Industrial Systems*, pages 43–48. ACM, 2015.

[103] H. Rosling, A. Rosling Rönnlund, and O. Rosling. New Software Brings Statistics Beyond the Eye. *OECD Statistics, Knowledge and Policy: Key Indicators to Inform Decision Making*, pages 522–530, 2005.

[104] H. Rosling, A. Rosling Rönnlund, and O. Rosling. *Factfulness: Ten Reasons We're Wrong About the World – and Why Things Are Better Than You Think*. Flatiron Books, 2018.

[105] P. Runeson, M. Höst, A. Rainer, and B. Regnell. *Case Study Research in Software Engineering: Guidelines and Examples*. John Wiley & Sons, 2012.

[106] A. Sannö, A. E. Öberg, E. Flores-Garcia, and M. Jackson. Increasing the Impact of Industry–Academia Collaboration through Co-Production. *Technology Innovation Management Review*, 9(4), 2019.

[107] A. E. Scott. Automatic Preparation of Flow Chart Listings. *Journal of the ACM*, 5(1):57–66, 1958.

[108] M. Shahin, M. A. Babar, and L. Zhu. Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access*, 5:3909–3943, 2017.

[109] C. E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27, 1948.

[110] G. Smith. Ethical risks of pursuing participatory research as an industrial doctoral student. In *Multidisciplinary Digital Publishing Institute Proceedings*, volume 1, page 167, 2017.

[111] Sogeti, Capgemini, and HP. World Quality Report 2014-2015, 2015.

[112] H. Spieker, A. Gotlieb, D. Marijan, and M. Mossige. Reinforcement Learning for Automatic Test Case Prioritization and Selection in Continuous Integration. In *International Symposium on Software Testing and Analysis*. ACM, 2017.

[113] M. Staron. *Automotive Software Architectures: An Introduction*. Springer, 2017.

[114] K.-J. Stol, P. Ralph, and B. Fitzgerald. Grounded Theory in Software Engineering Research: a Critical Review and Guidelines. In *International Conference on Software Engineering*, pages 120–131. ACM, 2016.

[115] P. E. Strandberg. Software Test Data Visualization with Heatmaps – an Initial Survey. Technical report, MDH-MRTC-318/2017-1-SE, Mälardalen University, 2017.

[116] P. E. Strandberg. Automated System Level Software Testing of Networked Embedded Systems. Thesis for Degree of Licentiate of Engineering, Mälardalen University, 2018.

[117] P. E. Strandberg. Ethical Interviews in Software Engineering. In *International Symposium on Empirical Software Engineering and Measurement*. IEEE, 2019.

[118] P. E. Strandberg, W. Afzal, T. Ostrand, E. Weyuker, and D. Sundmark. Automated System Level Regression Test Prioritization in a Nutshell. *IEEE Software*, 34(1):1–10, April 2017.

[119] P. E. Strandberg, W. Afzal, and D. Sundmark. Decision Making and Visualizations Based on Test Results. In *International Symposium on Empirical Software Engineering and Measurement*. ACM/IEEE, 2018.

[120] P. E. Strandberg, E. P. Enoiu, W. Afzal, D. Sundmark, and R. Feldt. Instrument from: Test Results Communication – An Interview Study in the Embedded Software Industry. Technical report, https://doi.org/10.5281/zenodo.1189562, 2018.

[121] 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*, 7:46434–46453, 2019.

[122] P. E. Strandberg, M. Frasheri, and E. P. Enoiu. Ethical AI-Powered Regression Test Selection. In *International Conference On Artificial Intelligence Testing*. IEEE, 2021.

[123] P. E. Strandberg, T. J. Ostrand, E. J. Weyuker, W. Afzal, and D. Sundmark. Intermittently Failing Tests in the Embedded Systems Domain. In *International Symposium on Software Testing and Analysis*, page 337–348. ACM, 2020.

[124] P. E. Strandberg, T. J. Ostrand, E. J. Weyuker, D. Sundmark, and W. Afzal. Automated Test Mapping and Coverage for Network Topologies. In *International Symposium on Software Testing and Analysis*, pages 73–83. ACM, 2018.

[125] P. E. Strandberg, D. Sundmark, W. Afzal, T. J. Ostrand, and E. J. Weyuker. Experience Report: Automated System Level Regression Test Prioritization Using Multiple Factors. In *International Symposium on Software Reliability Engineering*, pages 12–23. IEEE, 2016.

[126] Swedish Code of Statues. Higher Education Act (1992: 1434) (Högskolelagen (1992: 1434)), 1992.

[127] Swedish Research Council. Forskningsetiska principer i humanistisk-samhällsvetenskaplig forskning. Technical report, 1996.

[128] S. Thorve, C. Sreshtha, and N. Meng. An Empirical Study of Flaky Tests in Android Apps. In *International Conference on Software Maintenance and Evolution*, pages 534–538. IEEE, 2018.

[129] J. R. Ullmann. Bit-vector algorithms for binary constraint satisfaction and subgraph isomorphism. *ACM Journal of Experimental Algorithmics*, 15:1–6, 2010.

[130] A. Vahabzadeh, A. M. Fard, and A. Mesbah. An Empirical Study of Bugs in Test Code. In *International Conference on Software Maintenance and Evolution*, pages 101–110. IEEE, 2015.

[131] E. van Veenendaal. Standard glossary of terms used in software testing. Technical Report Version 2.3, International Software Testing Qualifications Board (ISTQB), March 2014.

[132] F. Vermeulen. On Rigor and Relevance: Fostering Dialectic Progress in Management Research. *Academy of Management Journal*, 48(6):978–982, 2005.

[133] K. R. Walcott, M. L. Soffa, G. M. Kapfhammer, and R. S. Roos. Time-Aware Test Suite Prioritization. In *International Symposium on Software Testing and Analysis*, pages 1–12. ACM, 2006.

[134] E. J. Weyuker and T. J. Ostrand. Experiences with Academic-Industrial Collaboration on Empirical Studies of Software Systems. In *International Symposium on Software Reliability Engineering Workshops*, pages 164–168. IEEE, 2017.

[135] K. Wiklund, S. Eldh, D. Sundmark, and K. Lundqvist. Impediments for software test automation: A systematic literature review. *Wiley Journal of Software Testing, Verification and Reliability*, 27(8):e1639, 2017.

[136] R. J. Wilson. *Introduction to Graph Theory, 5th Edition*. Prentice-Hall, 2010.

[137] W. H. Wolf. Hardware-Software Co-Design of Embedded Systems. *Proceedings of the IEEE*, 82(7):967–989, 1994.

[138] S. Yoo and M. Harman. Regression testing minimization, selection and prioritization: a survey. *Wiley Journal of Software Testing, Verification and Reliability*, 22(2):67–120, 2012.

[139] R. Yuster and U. Zwick. Finding Even Cycles Even Faster. *SIAM Journal on Discrete Mathematics*, 10(2):209–222, 1997.

[140] T. Zhang, H. Jiang, X. Luo, and A. T. Chan. A Literature Review of Research in Bug Resolution: Tasks, Challenges and Future Directions. *Oxford University Press: The Computer Journal*, 59(5):741–773, 2016.