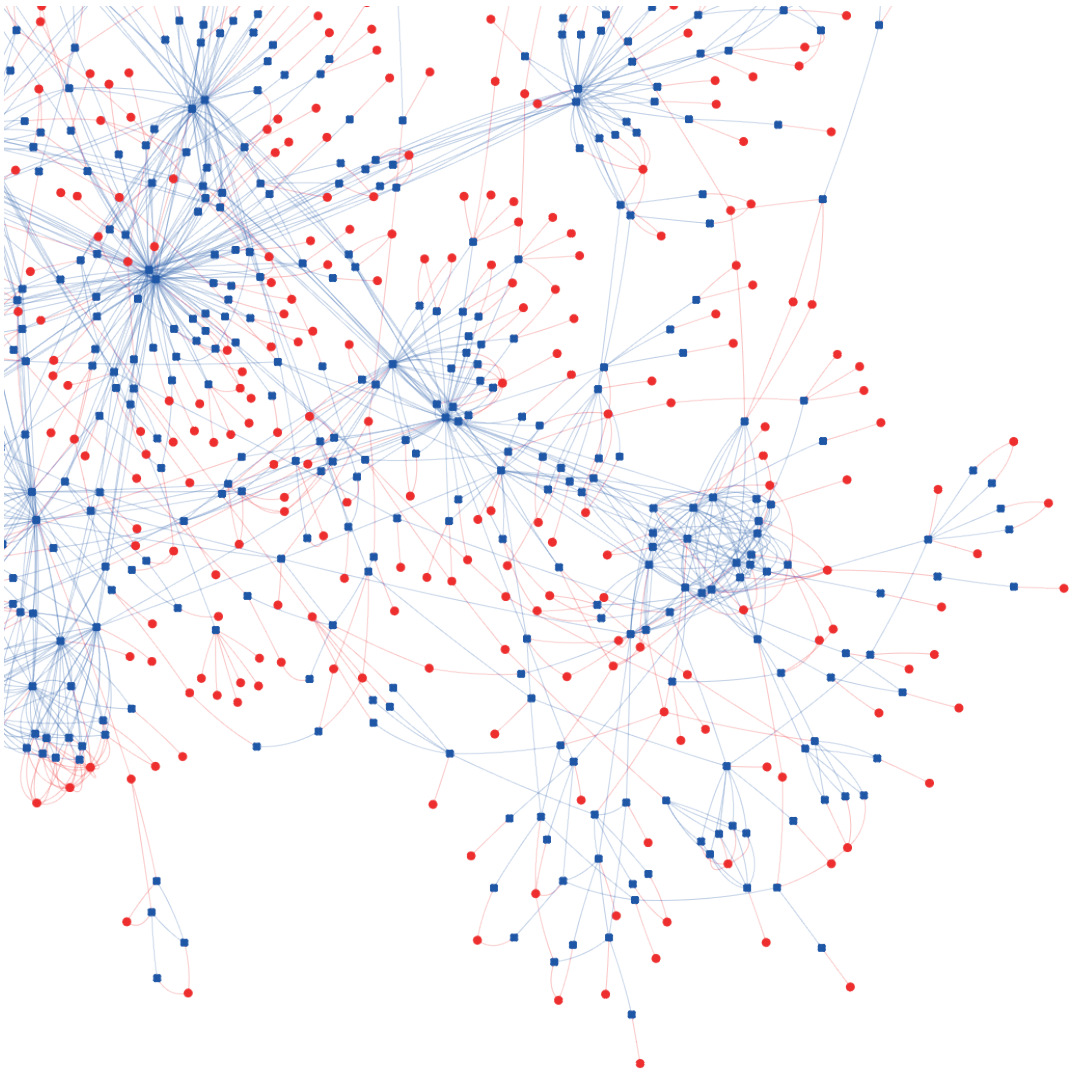
Mälardalen University Doctoral Dissertation 281

Multi-Criteria Optimization of System Integration Testing

Sahar Tahvili



MÄLARDALEN UNIVERSITY
SWEDEN

Mälardalen University Press Dissertations
No. 281

MULTI-CRITERIA OPTIMIZATION OF SYSTEM INTEGRATION TESTING

Sahar Tahvili

2018

School of Innovation, Design and Engineering

ISBN 978-91-7485-414-5
ISSN 1651-4238
Printed by E-Print AB, Stockholm, Sweden

Mälardalen University Press Dissertations
No. 281

MULTI-CRITERIA OPTIMIZATION OF SYSTEM INTEGRATION TESTING

Sahar Tahvili

Akademisk avhandling

som för avläggande av teknologie doktorsexamen i datavetenskap vid Akademin för innovation, design och teknik kommer att offentligen försvaras fredagen den 21 december 2018, 13.15 i Lambda, Mälardalens högskola, Västerås.

Fakultetsopponent: Professor Franz Wotawa, Graz University of Technology

Akademin för innovation, design och teknik

Abstract

Optimizing software testing process has received much attention over the last few decades. Test optimization is typically seen as a multi-criteria decision making problem. One aspect of test optimization involves test selection, prioritization and execution scheduling. Having an efficient test process can result in the satisfaction of many objectives such as cost and time minimization. It can also lead to on-time delivery and a better quality of the final software product. To achieve the goal of test efficiency, a set of criteria, having an impact on the test cases, need to be identified. The analysis of several industrial case studies and also state of the art in this thesis, indicate that the dependency between integration test cases is one such criterion, with a direct impact on the test execution results. Other criteria of interest include requirement coverage and test execution time. In this doctoral thesis, we introduce, apply and evaluate a set of approaches and tools for test execution optimization at industrial integration testing level in embedded software development. Furthermore, ESPRET (Estimation and Prediction of Execution Time) and sOrTES (Stochastic Optimizing of Test Case Scheduling) are our proposed supportive tools for predicting the execution time and the scheduling of manual integration test cases, respectively. All proposed methods and tools in this thesis, have been evaluated at industrial testing projects at Bombardier Transportation (BT) in Sweden. As a result of the scientific contributions made in this doctoral thesis, employing the proposed approaches has led to an improvement in terms of reducing redundant test execution failures of up to 40% with respect to the current test execution approach at BT. Moreover, an increase in the requirements coverage of up to 9.6% is observed at BT. In summary, the application of the proposed approaches in this doctoral thesis has shown to give considerable gains by optimizing test schedules in system integration testing of embedded software development.

ISBN 978-91-7485-414-5
ISSN 1651-4238

Abstract

Optimizing the software testing process has received much attention over the last few decades. Test optimization is typically seen as a multi-criteria decision making problem. One aspect of test optimization involves test selection, prioritization and execution scheduling. Having an efficient test process can result in the satisfaction of many objectives such as cost and time minimization. It can also lead to on-time delivery and better quality of the final software product. To achieve the goal of test efficiency, a set of criteria, having an impact on the test cases, needs to be identified. The analysis of several industrial case studies and also state of the art in this thesis, indicate that the dependency between integration test cases is one such criterion, which has a direct impact on test execution results. Other criteria of interest include requirement coverage and test execution time. In this doctoral thesis, we introduce, apply and evaluate a set of approaches and tools for test execution optimization at industrial integration testing level in embedded software development. Furthermore, ESPRET (Estimation and Prediction of Execution Time) and sOrTES (Stochastic Optimizing of Test Case Scheduling) are our proposed supportive tools for predicting the execution time and the scheduling of manual integration test cases, respectively. All proposed methods and tools in this thesis, have been evaluated at industrial testing projects at Bombardier Transportation (BT) in Sweden. As a result of the scientific contributions made in this doctoral thesis, employing the proposed approaches has led to an improvement in terms of reducing redundant test execution failures of up to 40% with respect to the current test execution approach at BT. Moreover, an increase in the requirements coverage of up to 9.6% is observed at BT. In summary, the application of the proposed approaches in this doctoral thesis has shown to give considerable gains by optimizing test schedules in system integration testing of embedded software development.

Keywords: Software Testing, Optimization, Integration Testing, Decision Support System, Dependency, Test Scheduling, Requirement Coverage

Sammanfattning

Optimering och förbättring av mjukvarutestningsprocessen har fått stor uppmärksamhet under de senaste årtiondena. Testoptimering är typiskt sett som ett multikriteriebeslutstödproblem. Aspekter av testoptimering innefattar testval, prioritering och schemaläggning. Att ha ett effektivt sätt för att köra olika testfall kan tillfredsställa många mål såsom kostnads- och tidsminimering. Det kan också leda till leverans i tid och bättre kvalitet på den slutliga mjukvaruprodukten. För att uppnå målet måste en uppsättning kriterier som har inverkan på testfallen identifieras. Analysen i denna avhandling av flera industriella fallstudier och toppmoderna metoder tyder på att beroendet mellan integrationstestfall är ett kritiskt kriterium, med en direkt inverkan på testresultaten. Andra viktiga kriterier är kravtäckning och testkörningstid. I denna doktorsavhandling introducerar vi, tillämpar och utvärderar en uppsättning metoder och verktyg för testoptimering på industriell integrationsnivå av mjukvara för inbäddade system. Dessutom, ESPRET (Estimation and Prediction of Execution Time) och sOrTES (Stochastic Optimizing Test case Scheduling) är våra stödjande verktyg för att förutsäga exekveringstid och för schemaläggning av manuell integrationstestning. Alla föreslagna metoder och verktyg i denna avhandling har utvärderats vid industriella testprojekt i Bombardier Transportation (BT) i Sverige. Slutligen har användning av de föreslagna metoderna i den här doktorsavhandlingen lett till förbättringar i form av en minskning av fel från redundanta testfall med upp till 40% jämfört med nuvarande metoder i BT, och en ökning av kravtäckning med upp till $9,6\%$.

Nyckelord: Programvarutestning, Optimering, Integrationstestning, Beslutsstödsystem, Beroende, Schemaläggning, Kravtäckning

Populärvetenskaplig sammanfattning

Rollen av mjukvara kan inte frånses från samhällets framsteg, med en direkt inverkan på våra dagliga liv. Att förbättra kvaliteten på mjukvaruprodukter har blivit allt viktigare för programvaruföretag under de senaste årtiondena. För att uppnå högkvalitativa mjukvaruprodukter måste man balansera ansträngningar mellan design och verifieringsaktiviteter under utvecklingsprocessen. Därför blir mjukvarutestning ett viktigt verktyg som bidrar till att tillgodose slutanvändarnas behov och att upprätthålla hög kvalitet på slutprodukten. Kvalitetssäkring av mjukvaruprodukter genererar stora satsningar på forskning inom programvarutestning.

Programvarutestning utförs manuellt eller automatiskt och övergången till automatiserade testningssystem har snabbt blivit utbredd i branschen. Eftersom automatiserad testning idag inte fullt ut kan dra nytta av mänsklig intuition, induktivt resonemang och inferens så spelar manuell testning fortfarande en viktig roll. Testning utförs ofta på flera nivåer, såsom enhet, integration, system och acceptans.

Lämplig testmetod (antingen manuell eller automatisk) beror på flera parametrar såsom kvalitetskrav, produktens storlek och komplexitet och testnivå.

Integrationstestning är den nivå i testprocessen där olika enskilda programmoduler kombineras och testas som en grupp och kan ofta vara den mest komplexa nivån. Integrationstestning utförs vanligtvis efter enhetstestning, när alla moduler har testats och godkänts separat.

För att testa en produkt manuellt måste en uppsättning testfallsspecifikationer skapas. En testfallsspecifikation beskriver textuellt en händelse och hur produkten ska uppträda vid angivna ingångsparametrar. Vanligtvis krävs en stor uppsättning testfall för att testa en produkt. Att köra alla testfall för en produkt

manuellt kräver tid och resurser. Därför har urval, prioritering och schemaläggning av tester fått stor uppmärksamhet i programvarutestningsdomänen.

I denna doktorsavhandling föreslår vi några optimeringstekniker för urval, prioritering och schemaläggning av manuella testfall för utförande. Alla föreslagna optimeringsmetoder i denna avhandling har utvärderats på industriella testprojekt vid Bombardier Transportation (BT) i Sverige.

There's just one life to live, there's no time to wait, to waste.
Josh Alexander

Acknowledgments

I would like to express my sincere gratitude to my main supervisor Markus Bohlin for the continuous support of my Ph.D. studies and related research, for his patience, motivation, and immense knowledge. A very special thank you goes out to my assistant supervisors Wasif Afzal, his guidance helped me for the entire duration of research and writing of this thesis and also Mehrdad Saadatmand for all his help and support. I have learned so much from all of you both personally and professionally, working with you made me grow as a researcher.

I am very grateful to my former supervisors Daniel Sundmark, Stig Larsson, Sergei Silvestrov, Tofigh Allahviranloo and Jonas Biteus, I have been extremely lucky to have supervisors who cared so much about my work and responded to my questions and queries promptly.

I would also like to thank my additional co-authors Leo Hatvani, Rita Pimentel, Michael Felderer, my master thesis students Sharvathul Hasan Ameerjan, Marcus Ahlberg and Eric Fornander working with you is a great pleasure, and also thanks to Narsis Aftab Kiani at Karolinska Institute and Mohammad Mehrabi for brainstorming and effective discussions.

My deepest gratitude goes to my family Mohammad, Shabnam, Saeed, Sara and Sepeher Tahvili and my friends: Shahab Darvish, Neda Kazemie, Lotta Karlsson, Jonas Österberg, Linnea Siem who have always been there for me no matter what. Without them I could have never reached this far.

I am thankful to Razieh Matini, Iraj and Siavash Mesdaghi, I consider myself extremely blessed to be a part of your family. There is no way I could ever thank you enough for being my second family in Sweden.

My sincere thanks also go to my manager, Helena Jerregard, who has always supported me throughout the work on this thesis. RISE SICS is a great workplace that I very much enjoy being part of. Furthermore, thanks to all my

colleagues at RISE SICS Västerås: Malin Rosqvist, Petra Edoff, Zohreh Ranjbar, Pasqualina Potena, Linnéa Svenman Wiker, Björn Löfvendahl, Daniella Magnusson, Markus Borg, Gunnar Widforss, Tomas Olsson, Kristian Sandström, Anders Wikström, Stefan Cedergren, Joakim Fröberg, Alvaro Aranda Munoz, Niclas Ericsson, Daniel Flemström, Martin Joborn, Cecilia Hyrén, Ksenija Komazec, Petter Wannerberg, Backer Sultan, Mats Tallfors, Peter Wallin, Thomas Nessen, Elsa Kosmack Vaara, Helena Junegard, Jawad Mustafa and also Barrett Michael Sauter for proofreading this doctoral thesis.

A special thanks to Ola Sellin, Stefan Persson, Kjell Bystedt, Anders Skytt, Johan Zetterqvist and the testing team at Bombardier Transportation, Västerås, Sweden.

In closing, I would like to express my sincere appreciation to Mariana Cook, a fine arts photographer with a kind heart, who granted the permission to use the image of Maryam Mirzakhani for the printed version of this doctoral thesis.

The work presented in this doctoral thesis has been funded by RISE SICS, ECSEL and VINNOVA (through projects MegaM@RT2, XIVT, TESTOMAT and IMPRINT) and also through the ITS-EASY program at Mälardalen University.

Sahar Tahvili
November 2018
Stockholm

List of Publications

Studies Included in the Doctoral Thesis[1]

Study A. *Dynamic Test Selection and Redundancy Avoidance Based on Test Case Dependencies*, **Sahar Tahvili**, Mehrdad Saadatmand, Stig Larsson, Wasif Afzal, Markus Bohlin and Daniel Sundmark, *The 11th Workshop on Testing: Academia-Industry Collaboration, Practice and Research Techniques (TAIC PART'16)*, 2016, IEEE.

Study B. *Cost-Benefit Analysis of Using Dependency Knowledge at Integration Testing*, **Sahar Tahvili**, Markus Bohlin, Mehrdad Saadatmand, Stig Larsson, Wasif Afzal and Daniel Sundmark, *The 17th International Conference on Product-Focused Software Process Improvement (PROFES'16)*, 2016, Springer.

Study C. *ESPRET: a Tool for Execution Time Estimation of Manual Test Cases*, **Sahar Tahvili**, Wasif Afzal, Mehrdad Saadatmand, Markus Bohlin and Sharvatul Hasan Ameerjan, *Journal of Systems and Software (JSS)*, volume 146, pages 26-41, 2018, Elsevier.

Study D. *Functional Dependency Detection for Integration Test Cases*, **Sahar Tahvili**, Marcus Ahlberg, Eric Fornander, Wasif Afzal, Mehrdad Saadatmand and Markus Bohlin, *Companion of the 18th IEEE International Conference on Software Quality, Reliability, and Security (QRS'18)*, 2018, IEEE.

[1] The included studies have been reformatted to comply with the doctoral thesis layout and minor typos have been corrected and marked accordingly.

Study E. *Automated Functional Dependency Detection Between Test Cases Using Text Semantic Similarity*, **Sahar Tahvili**, Leo Hatvani, Michael Felderer, Wasif Afzal and Markus Bohlin, Submitted to *The 12th IEEE International Conference on Software Testing, Verification and Validation (ICST'19)*, 2019, IEEE.

Study F. *sOrTES: A Supportive Tool for Stochastic Scheduling of Manual Integration Test Cases*, **Sahar Tahvili**, Rita Pimentel, Wasif Afzal, Marcus Ahlberg, Eric Fornande, Markus Bohlin, *Journal of IEEE Access*, 2018, In revision.

Additional Peer-Reviewed Publications, not Included in the Doctoral Thesis

Licentiate Thesis

1. *A Decision Support System for Integration Test Selection*, **Sahar Tahvili**, *Licentiate Thesis*, ISSN 1651-9256, ISBN 978-91-7485-282-0, October 2016, Mälardalen University.

Journal

1. *On the global solution of a fuzzy linear system*, Tofigh Allahviranloo, Arjan Skuka, **Sahar Tahvili**, *Journal of Fuzzy Set Valued Analysis*, volume 14, pages 1-8, 2014, ISPACS.

Conferences & Workshops

1. *Solving complex maintenance planning optimization problems using stochastic simulation and multi-criteria fuzzy decision making*, **Sahar Tahvili**, Sergei Silvestrov, Jonas Österberg , Jonas Biteus, *The 10th International Conference on Mathematical Problems in Engineering, Aerospace and Sciences (ICNPAA'14)*, 2014, AIP.

2. *A Fuzzy Decision Support Approach for Model-Based Trade-off Analysis of Non-Functional Requirements*, Mehrdad Saadatmand, **Sahar**

Tahvili, *The 12th International Conference on Information Technology: New Generations (ITNG'15)*, 2015, IEEE.

3. *Multi-Criteria Test Case Prioritization Using Fuzzy Analytic Hierarchy Process*, **Sahar Tahvili**, Mehrdad Saadatmand, Markus Bohlin, *The 10th International Conference on Software Engineering Advance (ICSEA'15)*, 2015, IARIA.

4. *Towards Earlier Fault Detection by Value-Driven Prioritization of Test Cases Using Fuzzy TOPSIS*, **Sahar Tahvili**, Wasif Afzal, Mehrdad Saadatmand, Markus Bohlin, Daniel Sundmark, Stig Larsson, *The 13th International Conference on Information Technology : New Generations (ITNG'16)*, 2016, Springer.

5. *An Online Decision Support Framework for Integration Test Selection and Prioritization (Doctoral Symposium)*, **Sahar Tahvili**, *The 25th International Symposium on Software Testing and Analysis (ISSTA' 16)*, 2016, ACM.

6. *Towards Execution Time Prediction for Test Cases from Test Specification*, **Sahar Tahvili**, Mehrdad Saadatmand, Markus Bohlin, Wasif Afzal, Sharvathul Hasan Ameerjan, *The 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA'17)*, 2017, IEEE.

7. *Cluster-Based Test Scheduling Strategies Using Semantic Relationships between Test Specifications*, **Sahar Tahvili**, Leo Hatvani, Michael Felderer, Wasif Afzal, Mehrdad Saadatmand, Markus Bohlin, *The 5th International Workshop on Requirements Engineering and Testing (RET'18)*, 2018, ACM.

Contents

I

Thesis

Chapter 1

Introduction

The role of software is important in our daily lives and to the progress of society in general. Over the past few decades, improving the quality of software products has become a unique selling point for software companies. Achieving high quality software products is possible through a balanced integrative approach of design and verification activities during the software development life cycle (SDLC) process [2]. Considering these facts, software testing becomes a major player in the product development process, which can satisfy the end users' needs and also ensure high quality of the final product [2].

Software testing research faces many challenges such as test effectiveness [3], where in this regard, the concept and nature of software testing has changed. The transition from manual to automated testing and continuous changes in the testing procedure have quickly become widespread at industry. However, human intuition, inductive reasoning and inference cannot be fully covered by the current form of test automation and therefore manual testing still plays a vital role in software testing [4].

Achieving a more effective testing process often comes down to dividing it into several levels, such as unit, integration, system and acceptance testing levels. Proposing an appropriate testing procedure (either manual or automated) depends on several parameters such as the quality of requirements, the size and complexity of the product and also the testing level.

Integration testing can be considered as the most complex testing phase in some practical scenarios. Integrating unit test modules and testing their behavior can result in a huge increase in complexity [5]. Verification of interactions between software modules has the objective of recognizing the correctness for

several modules of a system under test at least once, which ultimately results in a more complicated testing process, compared with other testing levels such as unit and acceptance testing.

Having a large set of test cases for testing a product manually in such a complex testing level proves the need for optimization methods in the software testing domain [6]. Test optimization is a multi-faceted topic consisting of writing effective requirement specification, creating more effective test cases, executing a subset of test cases which are required for a product release, ranking test cases for execution, etc. [7].

In this doctoral thesis we investigate methods to optimize the manual integration testing process through reducing unnecessary redundant test execution failures. Moreover, the proposed optimization approaches in this thesis lead to increased requirement coverage in each testing cycle. While several works advocate test optimization in the software testing domain [8], [9], to the best of our knowledge, this is the first attempt to provide an automated supportive tool which schedules manual integration test cases for execution stochastically. Moreover, most related to our work is the work by Nardo et al. [10], Elbaum et al. [11], Yoo and Harman [7], which address several approaches for test case selection, prioritization and scheduling.

In an efficient test optimization process, several factors such as test objective function, test constraint optimization and also the test case properties and features (e.g. execution time, requirement coverage) need to be identified in an early stage of testing. Testing time minimization is a crucial objective for test optimization, which is always demanded by industry as one of several optimization goals. Decreasing the total testing time can lead to on-time delivery of the final product and thereby leading to cost minimization. Furthermore, maximizing requirement coverage can be considered as another promising objective for the test optimization.

On the other hand, identifying and measuring the test case properties in advance is a challenging task, especially for manual integration test cases, which are usually described by testers in written text and are recorded together in a test specification document. However, since the testing constraints (e.g. allocated testing budget, testing deadline, delivery plan) need to be satisfied by one of the proposed test optimization approaches [12], the testing constraints should be determined implicitly before applying the test optimizations techniques. To address all of the above-mentioned optimization aspects, the test optimization problem must convert to a multiple-criteria decision making (MCDM) problem, where each test case property represents a criterion in the optimization model. In this thesis, we propose several MCDM methods for solving the test optimization

problem in the following forms: test case selection, prioritization and test scheduling. All of the proposed approaches in this thesis are applied and evaluated on a set of industrial testing projects at Bombardier Transportation (BT), a large railway equipment manufacturer in Sweden. In summary, applying the proposed optimization approaches in this doctoral thesis has resulted in an increase in the requirements coverage of up to 9.6%, where simultaneously, an improvement in terms of minimizing redundant test execution failures of up to 40% is observed with respect to the current execution approach at BT.

1.1 Thesis Overview

This doctoral thesis consists of two main parts. The first part provides an introduction to the overall work, while Chapter 2 gives background information on the research conducted in this thesis in the area of software testing. Chapter 3 presents an overview of the research, which consists of the thesis goals, the research challenges and corresponding technical and industrial contributions, followed by the research methodology. Finally, Chapter 4 draws the final conclusions and presents an outlook on future work. A collection of six research studies (Study A-to-F) constitutes the second part of the thesis that describes the research results. A summary of the included research studies in this doctoral thesis is as follows:

Study A: *Dynamic Test Selection and Redundancy Avoidance Based on Test Case Dependencies*, **Sahar Tahvili**, Mehrdad Saadatmand, Stig Larsson, Wasif Afzal, Markus Bohlin and Daniel Sundmark, *The 11th Workshop on Testing: Academia-Industry Collaboration, Practice and Research Techniques (TAIC PART'16)*, 2016, IEEE.

Abstract: Prioritization, selection and minimization of test cases are well-known problems in software testing. Test case prioritization deals with the problem of ordering an existing set of test cases, typically with respect to the estimated likelihood of detecting faults. Test case selection addresses the problem of selecting a subset of an existing set of test cases, typically by discarding test cases that do not add any value in improving the quality of the software under test. Most existing approaches for test case prioritization and selection suffer from one or more drawbacks. For example, to a large extent, they utilize static analysis of code for that purpose, making them unfit for higher levels of testing such as integration testing. Moreover, they do not exploit the

possibility of dynamically changing the prioritization or selection of test cases based on the execution results of prior test cases. Such dynamic analysis allows for discarding test cases that do not need to be executed and are thus redundant. This paper proposes a generic method for prioritization and selection of test cases in integration testing that addresses the above issues. We also present the results of an industrial case study where initial evidence suggests the potential usefulness of our approach in testing a safety-critical train control management subsystem.

Statement of Contribution: The first author is the main contributor of the study focusing on test case selection and prioritization based on dependency between manual integration test cases and other testing criteria; co-authors helped in study design and in writing related work.

Study B: *Cost-Benefit Analysis of Using Dependency Knowledge at Integration Testing*, **Sahar Tahvili**, Markus Bohlin, Mehrdad Saadatmand, Stig Larsson, Wasif Afzal and Daniel Sundmark, *The 17th international conference on Product-Focused Software Process Improvement (PROFES'16)*, 2016, Springer.

Abstract: In software system development, testing can take considerable time and resources, and there are numerous examples in the literature of how to improve the testing process. In particular, methods for selection and prioritization of test cases can play a critical role in using testing resources efficiently. This paper focuses on the problem of selecting and ordering of integration-level test cases. Integration testing is performed to evaluate the correctness of several units in composition. Furthermore, for reasons of both effectiveness and safety, many embedded systems are still tested manually. To this end, we propose a process for ordering and selecting test cases based on the test results of previously executed test cases, which is supported by an online decision support system. To analyze the economic efficiency of such a system, a customized return on investment (ROI) metric tailored for system integration testing is introduced. Using data collected from the development process of a large-scale safety-critical embedded system, we perform Monte Carlo simulations to evaluate the expected ROI of three variants of the proposed new process. The results show that our proposed decision support system is beneficial in terms of ROI at system integration testing and thus qualifies as an important element in improving the integration testing process.

Statement of Contribution: The first author is the main contributor of

this study focusing on both theoretical and industrial results, with the other co-authors having academic and industrial advisory roles. The simulation part was primarily the contribution of the second author. The first author developed the models, the concept, and also conducted the industrial case study. However, the writing process was an iterative contribution of all authors.

Study C: *ESPRET: a Tool for Execution Time Estimation of Manual Test Cases*, **Sahar Tahvili**, Wasif Afzal, Mehrdad Saadatmand, Markus Bohlin and Sharvatul Hasan Ameerjan, *Journal of Systems and Software (JSS)*, volume 146, pages 26-41, 2018, Elsevier.

Abstract: Manual testing is still a predominant and important approach for validation of computer systems, particularly in certain domains such as safety-critical systems. Knowing the execution time of test cases is important when performing test scheduling, prioritization and progress monitoring. In this work, we present, apply and evaluate *ESPRET* (EStimation and PRediction of Execution Time) as our tool for estimating and predicting the execution time of manual test cases based on their test specifications. Our approach works by extracting timing information for various steps in manual test specification. This information is then used to estimate the maximum time for test steps that have textual specifications but have not been previously executed. As part of our approach, natural language parsing of the specifications is performed to identify word combinations to check whether existing timing information on various test steps is already available or not. Since executing test cases on the several machines may take varying amounts of time, a set of regression models are used to predict the actual execution time for test cases. Finally, an empirical evaluation of the approach and tool has been performed on a railway use case at Bombardier Transportation (BT) in Sweden.

Statement of Contribution: The first author is the main contributor of the study, with the co-authors having academic and industrial advisory roles. The first author developed the prediction models, cross validation, the concept, and performed the industrial and also the evaluation case study. The last author implemented ESPRET as an automated supportive tool.

Study D: *Functional Dependency Detection for Integration Test Cases*, Sahar Tahvili, Marcus Ahlberg, Eric Fornander, Wasif Afzal, Mehrdad Saadatmand and Markus Bohlin, *Companion of the 18th IEEE International Conference on Software Quality, Reliability, and Security (QRS'18)*, 2018, IEEE.

Abstract: This paper presents a natural language processing (NLP) based approach that, given software requirements specification, allows the functional dependency detection between integration test cases. We analyze a set of internal signals to the implemented modules for detecting dependencies between requirements and thereby identifying dependencies between test cases such that: module 2 depends on module 1 if an output internal signal from module 1 enters as an input internal signal to the module 2. Consequently, all requirements (and thereby test cases) for module 2 are dependent on all the designed requirements (and test cases) for module 1. The dependency information between requirements (and thus corresponding test cases) can be utilized for test case selection and prioritization. We have implemented our approach as a tool and the feasibility is evaluated through an industrial use case in the railway domain at Bombardier Transportation, Sweden.

Statement of Contribution: The first author is the main contributor of this paper, while the co-authors having academic and industrial advisory roles. The first author conducted the industrial case study, evaluation of the proposed approach and developed the models and concepts. The second and third authors implemented the dependency extractor algorithms.

Study E: *Automated Functional Dependency Detection Between Test Cases Using Text Semantic Similarity*, **Sahar Tahvili**, Leo Hatvani, Michael Felderer, Wasif Afzal and Markus Bohlin, Submitted to *12th IEEE International Conference on Software Testing, Verification and Validation (ICST'19)*, 2018, IEEE.

Abstract: Knowing about dependencies and similarities between test cases is beneficial for prioritizing them for cost-effective test execution. This holds especially true for the time consuming, manual execution of integration test cases written in natural language. Test case dependencies are typically derived from requirements and design artifacts. However, such artifacts are not always available, and the derivation process can be very time-consuming. In this paper, we propose, apply and evaluate a novel approach that derives test case similarities and functional dependencies directly from the test specification documents written in natural language, without requiring any other data source. The approach first applies a deep-learning based language processing method to detect text-semantic similarities between test cases and then groups them based on two clustering algorithms HDBSCAN and FCM. The correlation between test case text-semantic similarities and their functional dependencies is

evaluated in the context of an on-board train control system from Bombardier Transportation AB in Sweden. For this system, the functional dependencies between the test cases were previously derived and are, in this paper, compared against the results of the new approach. The results show that of the two evaluated clustering algorithms, HDBSCAN has better performance than FCM or a dummy classifier. The classification methods' results are of reasonable quality and especially useful from an industrial point of view. Finally, performing a random undersampling approach to correct the imbalanced data distribution results in an F1 Score of up to 75% and an accuracy of up to 80% when applying the HDBSCAN clustering algorithm.

Statement of Contribution: The first author is the main contributor of the study focusing on theoretical, experimental and industrial results. The simulation part was primarily the contribution of the second author. The first author also developed the models, the concept and performed the industrial case study. The other co-authors have academic and industrial advisory roles.

Study F: *sOrTES: A Supportive Tool for Stochastic Scheduling of Manual Integration Test Cases*, **Sahar Tahvili**, Rita Pimentel, Wasif Afzal, Marcus Ahlberg, Eric Fornande, Markus Bohlin, *Journal of IEEE Access (IEEE-Access)*, 2018, In revision.

Abstract: The main goal of software testing is to detect as many hidden bugs as possible in the final software product before release. Generally, a software product is tested through executing a set of test cases, which can be performed manually or automatically. The number of test cases which are required to test a software product depends on several parameters such as: the product type, size and complexity. Executing all test cases with no particular order can lead to waste of time and resources. Test optimization can provide a partial solution for saving time and resources which can lead to the final software product being released earlier. In this regard, test case selection, prioritization and scheduling can be considered as possible solutions for test optimization. Most of the companies do not provide direct support for ranking test cases on theirs own servers. In this paper we introduce, apply and evaluate sOrTES as our decision support system for manual integration of test scheduling. sOrTES is a Python based supportive tool which schedules manual integration test cases which are written in a natural language text. The feasibility of sOrTES is studied by an empirical evaluation which has been performed on a railway use-case at Bombardier Transportation in Sweden. The empirical evaluation indicates that

around 40% of testing failure can be avoided by using the proposed execution schedules by sOrTES, which leads to an increase in the requirements coverage of up to 9.6%.

Statement of Contribution: The first author is the main contributor of the study focusing on both theoretical and experimental results, with the other co-authors having academic and industrial advisory roles. The performance evaluation is performed by the first and second author. Moreover, the fourth and fifth authors implemented sOrTES as an automated supportive tool. The model, concept development and industrial case study is performed by the first author.

Chapter 2

Background

In this chapter we provide a brief overview of some required background information of software testing concepts which are central to this doctoral thesis. Moreover, in this chapter we introduce our proposed approaches and solutions for solving the test optimization problem in an industrial domain. The following sections constitute mainly of the concepts and terminologies utilized in this thesis.

2.1 Software Testing

Software testing is a time and resource consuming process among the verification and validation (V&V) activities and can be considered as one of the critical phases in all software development life cycles [13]. According to the reports from both academia and industry, the process of software testing can take of up to 50% of total development cost [14].

IEEE international standard (ISO/IEC/IEEE 29119-1) provides a formal definition of the software testing process as follows:

Definition 2.1. *Software testing is the process of analyzing a software item with the aim to detect the differences between existing and required conditions (hidden bugs) and also to evaluate the features of the software item [15].*

The process of software testing at industry is performed manually, automatically or semi-automatically, and each one has its own advantages and disadvantages. A largely automated testing procedure eliminates manual efforts,

which can lead to saving testing time in some scenarios. However, the development and maintenance of automated tests costs between 3 to 15 times higher compared to manual testing [16].

In the manual testing procedure, the testing process is led by testers who operate a set of test case specifications manually. The testing process continues until the expected behaviors are ensured.

Definition 2.2. *A test case specification textually describes the main purpose of a specific test through providing a step-by-step procedure for execution [17].*

Moreover, the required inputs, expected outputs and test executed results (pass/fail) are also specified in a test case specification. Table 2.1 represents an example of a manual test case specification for a safety critical system at Bombardier Transportation (BT), which consists of a test case description, an expected test result, test steps, a test case ID, corresponding requirements, etc.

Test case name:	Drive And Brake Functions	**Date:**	2018-01-20
Test case ID	**Test level (s)**	**Test Result**	**Comments**
3EST000231-2756	Sw/Hw Integration		
Test configuration			
TCMS baseline:1.2 VCS Platform 3.24			
Requirement(s)			
SRS-Line Voltage 07 SRS-Speed 2051			
Tester ID			
BR−1211			
Initial State			
No active cab			
Step	**Action**	**Reaction**	**Pass / Fail**
1	No passenger emergency brake activated in consist = F	Traction safe loop deenergized $= 1$	
2	Restore the passenger emergency brake handle in the remote consist	Traction safe loop deenergized $= 0$	
3	Ready to run from A1, Logged in as Driver, MSTO	"Start inhibit reason"	
4	Wait 20 seconds		
5	Activate isolation of Fire System	Start inhibit reason $= 0$	
6	Deactivate Isolation of Fire System	Start inhibit reason $= 72(8 + 64)$	
7	Clean up		

Table 2.1: A test case specification example from the safety-critical train control management system at Bombardier Transportation.

As we can see, the test case specification presented in Table 2.1 is designed for manually testing the interaction between line voltage and speed modules. In order to determine the total number of required test cases for testing a software product, several factors including the product size, complexity, testing maturity, testing procedure (manual/ automated) and the level of testing need to be analyzed. An industrial testing project usually requires a large number of test cases in the various testing levels and therefore the majority of the total budget

should be allocated towards the testing activities in the software development process [18], [14].

2.2 Integration Testing

Performing testing activities in several levels makes it possible to detect more faults in the software product and also to evaluate the system's compliance with the specified needs [5]. Moreover, dividing the testing activities into separate levels can provide some clues for identifying missing areas in the software that have not been tested yet. A typical software development life cycle (SDLC) model is founded on six different phases including (i) requirement gathering and analysis, (ii) design, (iii) implementation, (iv) testing, (v) development and finally, (vi) maintenance. Generally, the testing phase is broken down into four main levels which are *unit, integration, system and acceptance testing*. Figure 2.1 illustrates the mentioned phases and the testing level as a V-model of the software development life cycle.

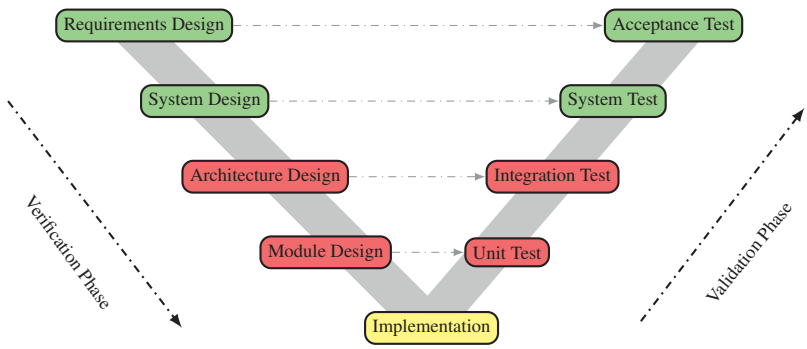


Figure 2.1: The V- Model for the software development life cycle.

Software testing is not only limited to the four mentioned levels but can also be performed in other types of testing such as regression testing [19], buddy testing and alpha-beta testing. These tests can take place at any of the four main levels with a specific purpose as well. Furthermore, some of the mentioned testing levels can be removed or merged with each other during a testing process while integration testing is performed in almost all testing projects in multi-component systems.

Definition 2.3. *Integration testing is a level of software testing which occurs after unit testing and before system testing, where individual modules are combined and tested as a group [5].*

In some testing scenarios, most of the hidden bugs in a software product can only be detected when the modules are interacting with each other [20], which makes the integration testing more complex [5].

2.3 Test Optimization

Today, test optimization and efficiency have become an increasingly popular topic in the software testing domain, and according to published research from academia, it is going to get even more important [21], [22]. As highlighted in Chapter 1, the testing process is a critical and costly process and therefore there is opportunity for an increase in test efficiency and a decrease in testing costs. There are a number of ways to optimize the testing process such as test suite minimization [23], test case selection, test case prioritization, test case scheduling and test automation. In this regard, a number of different algorithms have been applied to address the test optimization problem [24], such as ant colony optimization [25], particle swarm optimization [26], artificial bee colony optimization [27] and genetic algorithm [28]. In the following sections, we review some of the test optimization aspects which we then utilize for the optimization of industrial level integration testing.

2.3.1 Test Case Selection

Selecting and evaluating a subset of generated test cases for execution is one technique to optimize the testing process [29]. Rothermel and Harrold [30] formulate the test case selection problem as follows:

Definition 2.4. *Given: The program, P, the modified version of P, P', and a test suite, T.*
Problem: To find a subset of T, T', with which to test P'.

Test case selection can be considered as a proper optimization approach e.g. in exploratory and regression testing, where the behavior of a modified software can be verified through selecting a subset of test cases for re-execution [31]. Indeed, not all created test cases need to be executed at the same testing level as they can instead be tested in some other testing levels, for instance acceptance testing, where all test cases have already been executed at least once and only a

few test cases need to be selected for re-execution. Previously[1] in [32] and [33], we propose some methods of compensatory aggregation[2] for selecting a subset of test cases for execution.

2.3.2 Test Case Prioritization

To increase the rate of fault detection, all generated test cases should be ranked for execution in such a way that test cases of higher importance are ranked higher [35]. Test case prioritization can be applied almost at all testing levels with the main purpose of detecting faults earlier in the software product [7]. The problem of test case prioritization is defined by Rothermel and Harroldn [30] as:

Definition 2.5. *Given: A test suite, T, the set of permutations of T, PT and a function from PT to real numbers, $f : PT \rightarrow \mathbb{R}$.*
Problem: To find a $T' \in PT$ that maximizes f.

The main difference between Definition 2.4 and Definition 2.5 is the number of test cases. In Definition 2.4 a subset of test cases will be opted for the test case selection, whereas in Definition 2.5, all generated test cases will be ranked for execution in the test case prioritization. The problem of test selection and prioritization is addressed in this doctoral thesis in Studies A and B respectively.

2.3.3 Test Case Scheduling

Most of the previous works [36], [37], [29], [38] on test case selection and prioritization are only applicable before test execution, meaning that they do not monitor the test results after each execution. In order to optimize the testing process more efficiently, the test execution results of each test case need be recorded and considered continuously. Selecting and prioritizing test cases for execution based on their execution results leads us to utilizing the term *test case scheduling* as a proper technique for addressing the test optimization. In keeping with the structure of Definition 2.4 and Definition 2.5, we propose the following definition to the problem of test case scheduling:

Definition 2.6. *Given: A test suite, T. For all subset of T, $A \subseteq T$, the set of all permutations A, SPA. For all $B \subseteq T$, the set of all possible outputs after*

[1] Studies [32] and [33] are not included in this doctoral thesis.

[2] In a compensatory aggregation method, a set of alternatives and criteria need to be defined and identified respectively. The compensatory methods measure a weight for each criterion and also calculate the geometric distance between each alternative [34].

execution of the test cases in B, R. For each $r \in R$, the function $f_r : SPA \rightarrow \mathbb{R}$.
Problem: To find a prioritized set of T, T', considering the function $f_\emptyset : PT \rightarrow \mathbb{R}$, where PT is the set of permutations of T. To execute the test cases in T' until the first failure (if any). To update the previous procedure for $T - T_p$, considering the function f_{r_e}, until $T_p = T$, where T_p is the sets of passed test cases and r_e is the output of the executed test cases, respectively.

Indeed, the deciding factor for choosing test cases for execution and in what order, is dependent on the test execution results. Therefore, the executed test cases need to be saved in r_e and the prioritization process should be continued until all generated test cases have been executed at least once.

We need to consider that the main difference between Definition 2.4 and Definition 2.5 with Definition 2.6 is that only the last one monitors the results of the test executions, while the other two do not. This inclusion of results monitorization provides the opportunity for a dynamic test optimization process. If no failures occur after the first execution then we only need to prioritize test cases once, according to the Definition 2.5 (note that the f in Definition 2.5 is the same as $f_\emptyset$ in Definition 2.6).

2.4 Multiple-Criteria Decision-Making (MCDM)

The problem of test optimization can be categorized into a multi-criteria and also a multi-objective decision-making problem. The criterion in the software testing can be interpreted as a property for each test case which also help us to distinguish between test cases. Recognizing and measuring the criteria for each test case is a challenging task; first all properties cannot be determined precisely. Take for instance the test case property of *fault detection probability*, it can be utilized for test case selection and prioritization. There is no precise value for this property before the first execution but performing a historical analysis of a system under test (SUT) can provide some clues about the probability of detecting faults by each test case. In this regard, identifying most faulty subsystems (e.g. the brake system in a train contains more faults than the radio system) in the SUT can be used for comparing test cases with each other based on this property. For instance, test case A (which tests the brake system) has a higher probability of detecting faults than test case B (which tests the radio system); thus, test case A should be ranked higher for execution. Secondly, in some scenarios, measuring the test case properties requires being in close proximity to the testing experts at industries. To account for this, several

criteria have previously been proposed by researchers in the testing domain including *code coverage, test case size, execution time and cost, line of code and requirement coverage*. In this thesis, we define, utilize and measure the following criteria on each test case for addressing the test optimization issues in the form of test case selection, prioritization and scheduling for manual integration of test cases.

2.4.1 Requirement Coverage

Requirement coverage indicates the number of requirements which have been covered by each test case. The coverage of requirements is a fundamental necessity throughout the software development life cycle. In some scenarios, one test case can test more than one requirement and occasionally several test cases are assigned to test only a single requirement. Executing a test case with a greater requirement coverage (than other test cases) during the testing process can increase the value of requirement coverage in earlier stages. As part of this thesis we also propose an automated approach for measuring the requirement coverage for manual integration test cases (see Studies D and F).

2.4.2 Execution Time

As the title implies, test execution time is the total time that each test case requires for execution. Note that each execution of a test case can also result in a different execution time value. Knowing the execution time of test cases before execution is one way that test managers can divide test cases among several testers. Moreover, estimating the required time for each test case can provide a better overview of the total required time for testing a software product. In this doctoral thesis, we also introduce, apply and evaluate *ESPRET* as an automated tool for execution time estimation of manual integration test cases (see Study C).

2.4.3 Fault Detection Probability

It refers to the average probability of detecting a fault by each test case. Fault detection probability can be determined through performing historical analysis on the previously executed test cases. Sometimes, the fault detection probability is directly related to the complexity of the test cases. Selecting those test cases which have a higher chance for detecting the hidden faults in the system under

test can lead to earlier fault detection in each execution cycle. In this thesis, this criterion is measured by using a questionnaire-based study at BT (see Study A).

2.4.4 Test Case Dependencies

Previous studies have shown that executing test cases without considering the dependencies between them can cause failure at each time point during a testing process [39], [40]. The dependency issue becomes more visible in the integration testing level, where testing the interactions between modules can lead to a strong interdependency between the corresponding integration test cases. The dependent test cases directly influence the execution results of each other [39] and therefore the dependency issue can be considered as a critical problem in the integration testing levels. Indeed, paying no attention to the dependency between test cases can cause redundant test execution failures during the testing process. Moreover, the concept of dependency is important in a wide range of testing contexts (e.g. ranking test cases for execution, selecting a subset of test candidates for automation) and dependency detection has become a research challenge as well as an industrial challenge today [41]. There are several kinds of dependencies between test cases which have been identified by researchers, such as functional dependency, temporal dependency, semantic dependency and also abstract dependency. In our collaboration with testers and engineers at Bombardier Transportation, the functional dependency between integration test cases was identified as one of the most critical types of dependency.

Definition 2.7. *Test cases TC_1 and TC_2 are functionally dependent if they are designed to test different parts of function F_1 or if they are testing the interaction between functions F_1 and F_2.*

For instance, given two functions F_1 and F_2 of the same system, let the function F_2 be allowed to execute if its required conditions are already enabled by function F_1. Thus, function F_2 is dependent on function F_1. In this thesis, we assume that *all test cases which are designed to test F_2 should be executed any time after the assigned test cases for testing F_1*. Note that, in some testing scenarios, it may be the case that just some of the corresponding test cases for testing function F_1 should be executed before the corresponding test cases for testing function F_2. For instance, the required conditions for testing function F_2 might be enabled after testing some (e.g. 90%) of the designed test cases for function F_1. However, this assumption needs to be relaxed in the future.

Detecting functional dependencies between test cases can lead to a more efficient use of testing resources by means of:

- avoiding redundant test executions,
- parallel execution of independent test cases,
- simultaneous execution of test cases that test the same functionality,
- any combination of the previous options.

The following three main approaches are proposed, applied and evaluated for detecting the functional dependencies between manual integration test cases in this doctoral thesis:

1. Questionnaire-based, participant observation + archival data: The data collection for detecting the functional dependencies was done using participant observation, questionnaire as well as taking help from archival data for finding the cause of test case failures. The test experts at BT answered a questionnaire where the test dependencies were identified based on requirements. A dependency graph is designed and proposed, which can help testers when prioritizing test cases for execution based on the dependencies between test cases (see Study A).

2. Signal analysis: A set of internal signals in the implemented software modules is analyzed for detecting the functional dependencies between requirements and thereby identifying dependencies between test cases such that: `module 2` depends on `module 1` if an `output` internal signal from `module 1` enters as an `input` internal signal to `module 2`. Consequently, all requirements (and thereby test cases) for `module 2` are dependent on all the designed requirements (and test cases) for `module 1` (see Studies D and F).

3. Deep learning-based natural language processing: Since test specifications are usually written in a natural text, a natural language processing-based approach can help the testers for detecting the dependencies between test cases. To aid in this, we propose to use the Doc2Vec[3] algorithm [42] that, given the test specifications, allows automatic detection of test case dependencies and converts each test case into a vector in n-dimensional space. These vectors

[3]Doc2Vec is used to generate representation vectors out of a document, regardless of its length (see Study E and [42]).

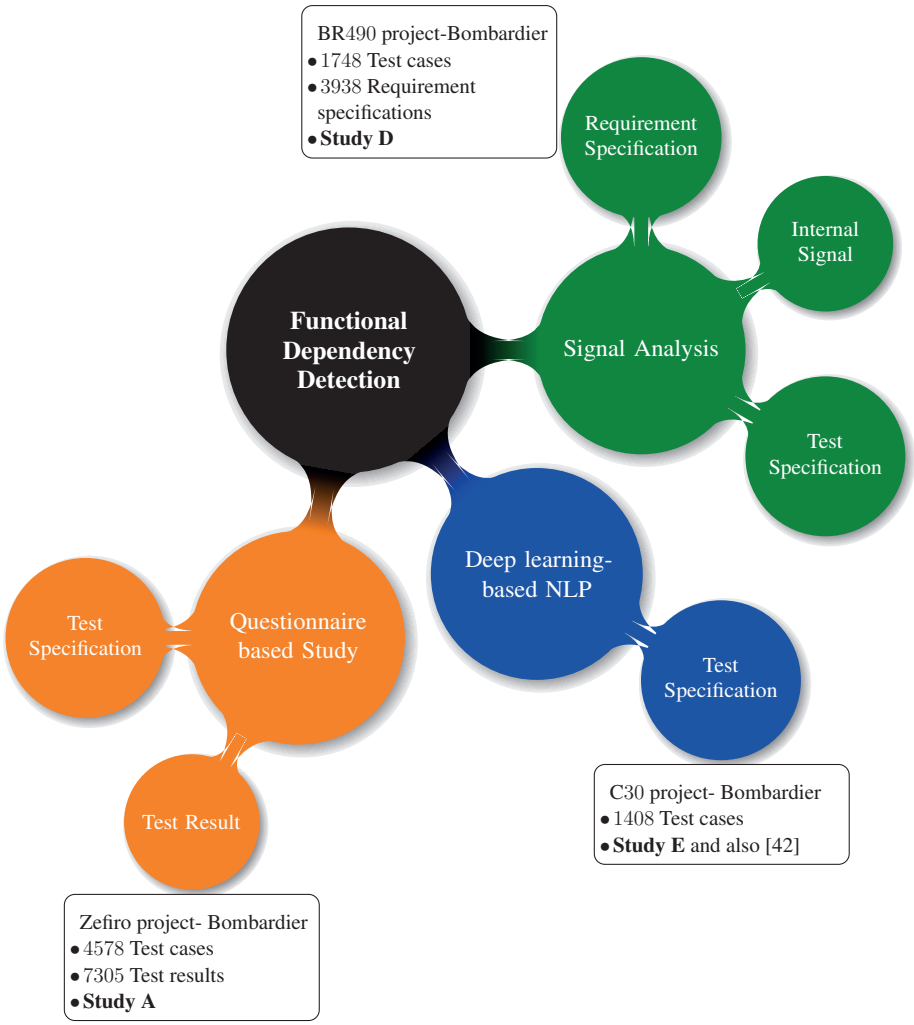


Figure 2.2: Three different approaches for functional dependency detection between manual integration test cases evaluated at Bombardier Transportation.

are then grouped using the HDBSCAN [4] clustering algorithm into several clus-

[4]Hierarchical Density-Based Spatial Clustering of Applications with Noise.

ters. Finally, a set of cluster-based test scheduling strategies are proposed for execution [42] (see Study E).

In order to show the feasibility of the proposed approaches, several industrial testing projects at BT have been selected and analyzed as use cases. However, the inputs for using the above approaches are different.

Figure 2.2 represents the proposed approaches for dependency detection, the required inputs for the main approaches, the name and size of the analyzed testing project at BT and also the corresponding completed study targeting each approach.

Chapter 3

Research Overview

In this chapter, we provide an overview of the doctoral thesis. First, we describe the research goals of the thesis. Later, the individual technical contributions which address the research goals are presented. Concluding the chapter is a comprehensive discussion of the research process and research methodology that is applied in this doctoral thesis.

3.1 Goal of the Thesis

In the past decade, it has become highly competitive among software companies to improve product quality [43], which in turn has also impacted the testing process [44]. Improving the quality of the software sometimes leads to increasing the final costs of the products [45]. Finding a trade-off between quality assurance and allocated testing resources at industry is a challenging issue and therefore optimizing the testing process has received much attention from researchers and also industry [46], [47].

This thesis aims to enable the applications of the test optimization techniques for improving the manual integration testing at industry. Under this target, we define our main research goal as follows:

To provide methods for a more efficient manual integration testing process, while decreasing unnecessary test execution failures and increasing the requirement coverage at each testing cycle.

This doctoral thesis is built on the prequel licentiate thesis titled: *A Decision Support System for Integration Test Selection* (see [48]), where a manual

multi-criteria decision support system (DSS) for test case selection and prioritization was proposed by us and the performance of the proposed DSS was evaluated at Bombardier Transportation through measuring the value of return on investment (ROI). The economic models presented in the licentiate thesis (included as Study B also in this doctoral thesis) confirms that there is a need for an automated decision support system for test case selection, prioritization and also scheduling at industry. While the licentiate thesis mainly focused on the design and economical evaluation of the proposed DSS, this doctoral thesis includes an automated version of the proposed DSS with industrial empirical evaluations.

As mentioned in Chapter 2 and also in the licentiate thesis [48], executing all generated test cases without any particular order may cause unnecessary test execution failures and thereby leads to a waste of testing resources and time. On the other hand, applying manual approaches for optimizing the testing process is a time and cost consuming procedure and requires even more testing resources [48]. In the licentiate thesis, the test case properties (criteria) had been measured through performing a set of questionnaire-based studies, which can be a taxing process and therefore prone to human error. Table 3.1 shows a sample survey that had been sent to the testing experts at BT for measuring the test case properties.

Test Case ID	Execution Time	Requirement Coverage	Dependent on	Execution Cost	Fault Detection Probability
Drive-S-046	VH	H	Air-S-005, Brake-S-25	M	H
Speed-S-005	M	VL	Speed-S-21, Voltage-IVV-4	M	L
Doors-S-011	VL	H	Brake-S-65, Air-S-005, Drive-S-09	L	H
Doors-S-022	H	L	Battery-S-13, DVS-IVV-08	L	L
Brake-IVV-31	VL	M	None	L	M
Brake-IVV-41	VL	L	Fire-IVV-125	M	M
Drive-S-024	L	H	Speed-IVV-66, Battery-S-58	M	H
Speed-IVV-04	L	M	HVAC-S-06, Speed-S-17, Brake-S-02	M	M
Drive-IVV-30	L	H	None	L	M
Brake-S-044	L	H	None	H	M
Brake-S-042	VL	H	Bogies-IVV-225, Brake-S-88	M	M
Drive-S-011	VL	M	Radio-S-25, Front-S-002	H	M

Table 3.1: A sample survey with values very low (VL), low (L), medium (M), high (H) and very high (VH), utilized for measuring five test case properties (execution time, requirement coverage, dependency, execution cost and fault detection probability) at BT in the licentiate thesis [48]. The highlighted columns represent those properties which are measured automatically today.

Table 3.1 represents the testers' opinions about five test properties captured

by using a set of fuzzy linguistic variables[1] (very low, low, high, etc.). However, we were faced with several situations where testers had different opinions about each property of the test cases.

In this doctoral thesis, the following test case properties are measured automatically: *requirement coverage, execution time* and *the dependencies between test cases*, which are also highlighted in Table 3.1.

Moreover, test cases have been semi-automatically prioritized for execution by using two compensatory aggregation methods AHP[2] and TOPSIS[3] in the licentiate thesis. As discussed previously, in order to continuously optimize the testing process the execution results of each test case need be recorded and analyzed, which requires using automated tools. For this reason, we have extended the scope of our research on automation issues since the publication of the licentiate thesis. Thus, in this doctoral thesis, we propose two automated supportive tools which are empirically evaluated on several large-scale testing projects at Bombardier Transportation.

By considering a large number of manual integration test cases and their property measurements, together with dynamic decision making based on test case execution results, the research goals can be defined as follows:

RG1: Defining approaches for automatically detecting dependencies between test cases.

Motivation: The dependencies between test cases have been identified as one of the most important criteria for test optimization at the integration testing level [50], [39], [51]. Paying no attention to the dependencies between integration test cases might lead to sequential failure of test cases [39], [52] and thereby a waste of testing resources. On the other hand, the manual approaches for dependency detection are costly approaches and suffer from uncertainty. Since our final goal in this thesis is to address test efficiency, we focus on semi and fully automated approaches for dependency detection between manual integration test cases.

RG2: Automatically estimating the execution time for manual test cases.

Motivation: Knowing the execution time of test cases in an early stage of a testing process can help testers and test managers to select, prioritize and schedule test cases for execution. Creating a database of the previously

[1] In a fuzzy partition, every fuzzy set corresponds to a linguistic concept such as very low, low, average, high, very high [49].

[2] Analytic Hierarchy Process.

[3] Technique for Order of Preference by Similarity to Ideal Solution.

executed test cases and performing various regression analysis techniques may provide clues to estimate the required time needed for executing new generated test cases on the same system under test. A supporting tool can be connected to a database to predict the execution time for the new test cases. This research goal addresses the automation of test case execution time estimation.

RG3: Automatically measuring the requirement coverage for manual test cases.

Motivation: Comparing and measuring the number of allocated requirements to each test case can be considered as a solution for ranking test cases for execution. Comparing and ordering test cases for execution based on their requirement coverage can lead to the testing of more requirements with overall fewer test cases which thereby minimizes the total testing time. However, this information is not always available in test case specifications and sometimes we need to analyze the requirement specifications for measuring this value. This research goal addresses the automation of requirement coverage measurements.

RG4: Proposing automated (and semi-automated) methods for test case selection, prioritization and scheduling.

Motivation: Meeting research goals 1 and 2 can help us to optimize the execution orders of test cases based on their dependencies and also execution time. Moreover, an automated approach in the form of a decision support system can be utilized as a supportive tool for test case selection, prioritization and scheduling, which also monitors the results of each test execution.

RG5: Measuring the effectiveness in terms of cost and time reduction of using the proposed optimization approaches.

Motivation: Evaluation of feasibility and efficiency of the proposed optimization approaches in terms of reducing redundant tests execution failures and increasing the requirement coverage at each testing cycle. Applying the proposed approaches in this thesis can lead to decreasing required troubleshooting time and thereby minimizing testing cost. The usage of the proposed optimization approaches can also lead to increasing the value of return on investment (ROI) for the testing companies.

Figure 3.1 illustrates a holistic overview of six performed studies (A-to-F), which supports the main objective of this PhD thesis. Moreover, the performed

studies provide contributions to the body of knowledge in the field of test optimization.

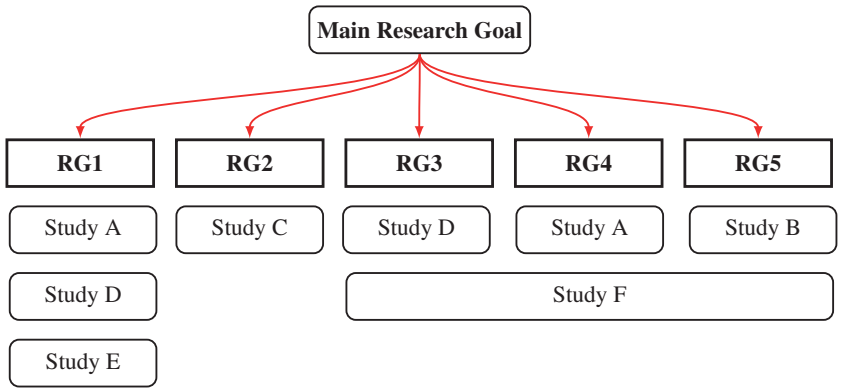


Figure 3.1: Holistic overview of how the studies included in this PhD thesis support the research goals.

3.2 Technical Contributions

This section provides an overview of the contributions included in this doctoral thesis. The next section describes a high-level overview of the provided contributions and also how they together achieve the main goal of test efficiency. Later in this section, we discuss individual contributions in detail.

3.2.1 Overview of the Proposed Approach

In order to realize the main goal of this doctoral thesis, we proposed, applied and evaluated approaches for improving the manual integration testing process through gathering the empirical evidence, both for or against, the use of proposed optimization methods in industrial practice.

3.2.2 Discussion of Individual Contributions

The technical contributions presented in this thesis can be categorized into five main contributions:

C1: Recognizing the challenges of test optimization
Software quality is playing a more important role than ever before and therefore an increase in cost in the software product itself is expected [53]. The increasing demand for quality, fast turnover, and limitations in resources have encouraged researchers to provide and utilize optimization techniques in the software testing domain [54], [55], [56]. Test case selection, prioritization and scheduling have been identified as potential approaches for optimizing the testing process. Previously in [33] and [32], we proposed two methods of compensatory aggregation for addressing the mentioned issues, where test cases are selected and prioritized for execution based on their properties. Organizing test cases for execution without paying attention to their execution results might yield less than an optimal outcome. In fact, the process of test case selection and prioritization should be performed dynamically until all test cases are executed successfully (**challenge 1**). Moreover, the problem of test optimization is a multi-criteria decision making problem (**challenge 2**). However, identifying and measuring the test case properties (criteria) is a challenge being taken on by researchers, one of the reasons being that it requires a close proximity to industry. The following test case properties have been recognized by us as critical properties, which directly impact the process of test optimization: dependency between test cases, requirement coverage, execution time and cost and fault detection probability. Measuring the effects of the mentioned properties automatically for each test case is a research challenge, especially in a manual testing procedure, where test cases are written by testers in a natural language (**challenge 3**).

Targeting Research Goal: RG3, RG4
Included Study: Study A, D, F

C2: Execution time prediction
As discussed in **challenge 2**, proposing an automated way for measuring test case execution time is required. The increased complexity of today's test optimization, together with the increased number of test cases that are created during a testing process, require prediction models for execution time prediction. We investigate ESPRET[4] as an automated tool for execution time estimation and prediction of manual integration test cases.

Targeting Research Goal: RG2
Included Study: Study C

[4] EStimation and PRediction of Execution Time.

C3: Requirement coverage measurement
Through performing a pairwise comparison of identified test case properties (criteria), we came to the realization that the requirement coverage has the highest priority (around 67.5%) for test case selection and prioritization at BT [33], [57]. In this thesis, we propose an automated approach for measuring the number of requirements covered by each test case.
Targeting Research Goal: RG3
Included Study: Study D, F

C4: Dependency detection
Our studies indicate that integration test cases can fail based on four main reasons: (i) there is a mismatch between test case and its corresponding requirement, (ii) the testing environment is not yet adequate for testing efficiently, (iii) there exist bugs in the system which is under test and (iv) no attention is paid to the dependencies between test cases. Among the mentioned causes, failures based on failures between interdependent test cases are preventable. As illustrated in Figure 2.2, the dependency between manual integration test cases is detected through applying three different approaches in this doctoral thesis.

Targeting Research Goal: RG1
Included Study: Study A, D, E

C5: An automated decision support system and measures of effectiveness
Using manual or semi-automated approaches for selecting, prioritizing or scheduling test cases for execution requires testing resources and human judgment and suffers from uncertainty and ambiguity. Executing a large set of test cases several times during a testing project is difficult to handle manually. An automated decision support system (DSS) can analyze the test cases and make decisions for execution orders of test cases more easily. However, the mentioned test case properties can be measured automatically inside the DSS. Some of the properties (e.g. dependency between test cases) can be changed after execution, and therefore the properties should be re-measured after each test execution. Optimizing the execution orders of test cases based on their execution results can provide a more efficient way of using testing resources. In this doctoral thesis, we investigate sOrTES[5]as a supportive tool for stochastic scheduling of manual integration test cases. Additionally, sOrTES is able to measure the requirement coverage for each test case and also detects

[5]Stochastic Optimizing of TEst case Scheduling.

the dependencies between manual integration test cases. Moreover, the effectiveness of sOrTES and manual and semi-automated approaches in this thesis is measured in terms of maximizing the return on investment through applying the approaches at BT.

Targeting Research Goal: RG5
Included Study: Study F, B

3.3 Research Process and Methodology

Industrial research includes more than just publishing research results or technical reports [58]. In fact, it requires a close cooperation between industry and academia during the entire research process, where the academic research results can be evaluated in a real industrial setting and thereby improve the industrial development process [59]. A close and dynamic collaboration between researchers and practitioners is the golden key to success [60]. Additionally, choosing a proper research strategy can help the researchers to achieve valid answers to their research questions. The research conducted in this doctoral thesis utilized case studies[6] and various data collection strategies (unstructured interviews, document analysis and observation). In summary the research methodology used in this research is described in the following process:

(i) Identifying the research problems and challenges through reviewing the state of the art, current issues and observation and also employing semi-structured interviews with the testing experts at Bombardier Transportation in Sweden.

(ii) Selecting challenges to solve and designing the research objectives, goals and questions.

(iii) Proposing a set of solutions for the identified research goals.

(iv) Evaluating the proposed solutions with testing experts at Bombardier Transportation by running simulations of illustrative examples and also conducting empirical evaluations.

The structure of the research process and framework can be summarized as depicted in Figure 3.2, which has been adapted from the proposed technology transfer model by Gorschek et.al in [58].

[6]The provided industrial case studies in this doctoral thesis are following the proposed guidelines for conducting and reporting case study research in software engineering by Runeson and Höst [61].

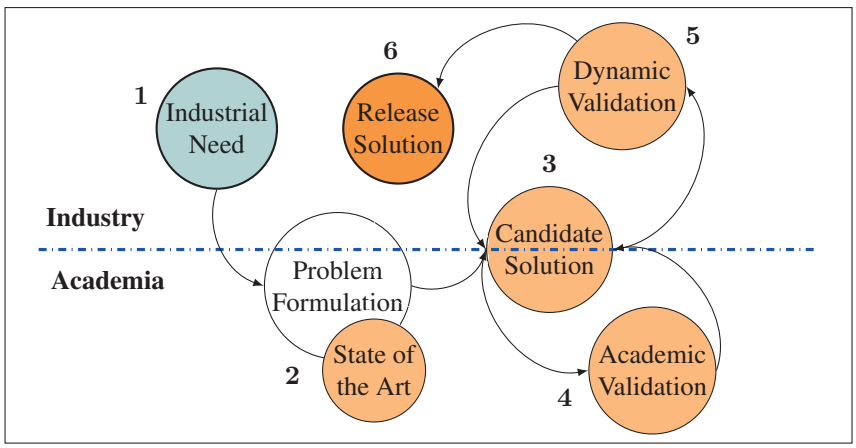


Figure 3.2: The research model and technology transfer overview used in this doctoral thesis, adapted from Gorschek et.al [58].

As can be seen by Figure 3.2, the proposed research model by Gorschek et.al [58] is divided into two main phases: *industry and academia*, where both phases are dynamically communicating with each other throughout six individual steps. We now present how the technology transfer model outlined in Figure 3.2 is applied in this doctoral thesis.

- **Step 1. Identify potential improvement areas based on industry needs**

  It is critical to consider the demands of the industry before designing research questions [60]. Thus, we started our research by observing an industrial setting. During this process, several potential areas of improvement at various testing levels at BT were identified, of which the integration testing level has been selected as a viable candidate for improvement. Moreover, the test optimization problem in the form of test case selection, prioritization and scheduling has been recognized as a real industrial challenge at BT during our process assessment and observation activities.

- **Step 2. Problem formulation**

  Based on the identified demands in the previous step and by collaborating closely with the testing experts at BT, the problem statement was formulated. The testing department at BT consists of several testing teams

including software developers, testers, testing team leaders and middle managers [48]. The researchers from academic partners in the several research projects (e.g. TESTOMAT[7], MegaM@RT2[8], IMPRINT[9] and TOCSYC[10] projects) have regular meetings with BT. Furthermore, a set of non-directive interviews were employed in this step, which established a common core and vocabulary of the research area and the system under test [58] between researchers and testing experts.

- **Step 3. Formulate candidate solutions**

  In a continuous collaboration with the testing teams at BT, a set of candidate solutions for improvement of the integration testing process were designed. In this step, BT covered the role of keeping the proposed solutions compatible with their testing environment [48]. On the other hand, we as the research partners took on the main responsibility for keeping track of state of the art in the test optimization domain and applying the proposed solutions with a combination of new ideas [58]. We designed a multi-criteria decision support system for execution test case scheduling. The main purpose of the proposed solution was to measure test case properties and schedule them for execution based on those properties. With an agreement with BT, the proposed solution was selected as the most promising solution for test optimization at the integration testing level at BT.

- **Step 4. Academic validation**

  In the principal technology transfer model proposed by Gorschek et.al in [58], several steps are considered for evaluating the candidate solutions proposed in the previous step. In this doctoral thesis, we employed academic and dynamic validation methods to evaluate the proposed solution for solving the test optimization problem at BT. Our scientific work was evaluated by international review committees from the venues where we published our research results, of which six studies have been selected and presented in this doctoral thesis (Study A-to-F). In this step the limitations of the various approaches are identified and certain solutions for addressing these limitations are provided as future work. Two initial

[7] The Next Level of Test Automation [62].

[8] An scalable model-based framework for continuous development and run-time validation of complex system [63].

[9] Innovative Model-Based Product Integration Testing [64].

[10] Testing of Critical System Characteristics [65].

prototype versions of the proposed approaches were implemented by our master thesis students in the software engineering program at Mälardalen University and also applied mathematics program at KTH Royal Institute of Technology.

- **Steps 5. Dynamic validation**

  This step has been performed through two research projects (TESTOMAT and MegaM@RT2). According to the project's plan, a physical weekly meeting needs to be held at BT between all industrial and academic partners who are involved in the mentioned research projects [48]. The results of the conducted case studies, prototypes and experiments are presented by researchers during the meetings. Moreover, some small workshops are organized by us for the team members of different internal testing projects (mostly BR490[11] and C30[12] projects) at BT. The industrial partners (testing expert at BT) gave valuable feedback, some of which are applied in this step. Tool support was the main feedback for the proposed solutions that we received from BT.

- **Step 6. Release the solution**

  After receiving and analyzing the feedback from the academic and dynamic validation steps, the proposed solutions are then implemented as actual supportive tools. The initial versions of the proposed solutions are then implemented by our master thesis students and BT engineers. In this regard, first ESPRET is released as a supportive tool for estimating and predicting the execution time of manual test cases based on their test specifications (see Study C). ESPRET is a Python based tool which helps testers and test managers to estimate the required execution time for running manual test cases and can be employed for test case selection, prioritization and scheduling. As outlined in Chapter 2, the execution time is one of the test case properties which needs to be identified in an early stage of the testing process. Today, this property is measured automatically at BT by using ESPRET, of which is highlighted in Table 3.1 as an automatically measured property. Secondly, sOrTES is implemented and released by us as an automated decision support system for stochastic scheduling of manual integration test cases for execution (see Study F).

[11] is an electric rail car specifically for the S-Bahn Hamburg GmbH network in production at Bombardier Hennigsdorf facility [66].

[12] is the project name of the new subway carriages ordered by Stockholm public transport in 2008 [67].

sOrTES consists of two separate phases: extractor and scheduler, where the dependencies between test cases and their requirements coverage are measured in the extractor phase (highlighted in Table 3.1 as an automatically measured property). The scheduler phase embedded in sOrTES ranks test cases for execution based on their dependencies, requirement coverage, execution time (extracted from ESPRET) and also test case execution results. Full implementation of the released solutions in this doctoral thesis is pending on results from the trial releases. However, our ultimate goal is to integrate ESPRET and sOrTES and to incrementally release it as an open source tool.

Chapter 4

Conclusions and Future Work

This chapter concludes the doctoral thesis. A summary of the main contributions is presented in Section 4.1, while Section 4.2 contains some suggestions for future research of this thesis.

4.1 Summary and Conclusion

The overall goal of this doctoral thesis is to provide methods for a more efficient manual integration testing process. To work towards this goal, we have conducted research in three sequential stages: (i) measuring the properties of test cases automatically, (ii) prioritizing and scheduling test cases for execution automatically in terms of a decision support system, and (iii) empirically evaluating the effectiveness of the proposed approach. All included publications in this doctoral thesis build on empirical research through performing several industrial case studies at Bombardier Transportation AB in Sweden. The measurement stage of the test case properties is presented by five studies. In Study A we were able to assess the test case properties based upon a questionnaire conducted on an industrial use case. Moreover, in Study A we show that *a fuzzy linguistic variable can be assigned to each test case as a property, however, this process is a time consuming process as it requires human judgment and assessment.* We also show that applying methods of compensatory aggregation can be utilized for ranking and prioritizing test cases for execution.

Study B reports an economic model in terms of return on investment (ROI) value for the proposed manual approach in Study A. Moreover, a semi and also a fully automated approach for data gathering, data analysis and decision making are simulated in Study B in order to a find a trade-off between effort and return. In Study B we show that *even performing a manual approach for ranking test cases for execution can reduce unnecessary test execution failures and yields a positive value for ROI. Furthermore, having a semi and a fully automated DSS leads to gaining additional value for ROI*. Since the publication of Study B we have extended the scope of our research on automation of the proposed approaches in Study A.

Three test case properties are selected for automation: execution time, dependency and requirement coverage. Moreover, providing an automated tool support which can schedule test cases for execution is also considered in this doctoral thesis. In this regard, in Study C we introduce, apply and evaluate ESPRET as a Python based supportive tool for estimating execution time for manual integration test cases. In Study C we show that *the execution time of manual test cases is predictable through parsing their test specifications and performing regression analysis techniques on the previously executed test cases*.

Study D provides an automated approach for functional dependency detection between manual test cases at the integration testing level. The necessary inputs for running the proposed approach in Study D are test case and requirement specifications. Study D indicates that *the signal communications between software modules provide some clues about the dependency between them and thereby the dependency between their corresponding requirements and test cases.*

Since the signal information and requirement specifications are not available for all systems under test, we proposed another automated approach for dependency detection which only requires the test case specifications. Study E shows that *the dependency between manual test cases can be detected by analyzing their test specifications using deep learning-based NLP techniques. However, the similarity between test cases can also be discovered by applying the proposed approach.*

Study F builds on Study D, where we propose sOrTES as an automated tool for stochastic scheduling of test cases. The proposed automated approach in Study D is embedded in sOrTES for dependency detection for those situations where the signal information and requirement specifications are available. In Study F we schedule test cases for execution based on their requirement coverage and also dependencies, where after each test execution, a new test schedule will be proposed based on the results of the previous execution of each test

case. Study F indicates that *the automatic scheduling of test cases based on their properties and execution results can reduce unnecessary redundant test execution failure by up to 40%, which leads to an increase in the requirements coverage of up to 9.6%.*

As mentioned earlier in this chapter, in order to show the feasibility of all proposed approaches in the doctoral thesis (Study A-to-F), several empirical evaluations have been performed on a railway use-case at Bombardier Transportation in Sweden.

4.2 Future Work

Work on this doctoral thesis has opened several future research directions for us:

- Future development of sOrTES and integration possibilities for merging ESPRET and sOrTES as one open source tool for integration test stochastic scheduling. Today, we predict the execution time for each test case using ESPRET and the results are inserted manually into the extractor phase of sOrTES. Embedding ESPRET into the extractor phase of sOrTES can help us to measure three test case properties automatically by using only a single tool.

- The proposed deep learning-based NLP approach for dependency detection in Study E can also be added to the extractor phase of sOrTES, where it captures the test case specifications as input and clusters the dependent test cases. Thus, sOrTES can be used in those situations where any of the test process artifacts (e.g. signal information, requirement specification, test specification) are available.

- As outlined in Table 3.1, there are two properties (execution cost and fault detection probability) of test cases which are still candidates for automation. Execution cost can be calculated by using the execution time of each test case. It can also be calculated by adding more parameters such as the complexity of the test case, the testing level and also the number of requirements which have been assigned to each test case. Moreover, the probability of detecting faults by each test case is also measurable via performing historical analysis on some previously tested project within the same testing environment. For instance, the brake system at BT is the most faulty system and thus test cases which are assigned to test the

brake system have a higher probability of detecting these hidden faults in the system compared with other test cases. Prioritizing the systems and the subsystems for implementation and testing can be considered as another possibility for contributing to the existing research.

- The concept of requirement coverage indicates the number of requirements assigned to be tested by each test case. In some testing scenarios, one requirement can be assigned to more than one test case. In all presented studies in this thesis, the number of allocated requirements to each test case are summed and presented as a numerical value. In the future, this value will be modified through combining with a *unique requirement coverage* value. A *unique requirement coverage* indicates the number of requirements which are only assigned to one test case. Considering this value in the test execution helps testers to test all requirements at least once in any particular proposed test execution order.

- The dependency between the functions can also be expressed as a ratio. For instance, the total dependency ratio for function F_1 and function F_2 is 90% (F_2 depends on F_1), meaning that, after 90% successful executions of the designed test cases for function F_1, we are able to test the assigned test cases to function F_2. Finding the dependency ratio between the functions can help us to relax the assumption that, *all test cases which are designed to test F_2 should be executed any time after the assigned test cases for testing F_1*. Indeed, some of the dependent test cases can be tested in parallel with other test cases, which leads to saving time in a testing cycle. Since the dependencies between software modules, functions, requirements and test cases are detected in this doctoral thesis, analyzing the test records on the previously executed dependent test cases might provide clues about the dependency ratio between dependent test cases.

Bibliography

[1] M. Cook and R. Gunning. *Mathematicians: An Outer View of the Inner World*. Princeton University Press, 2009.

[2] M. Young. *Software Testing and Analysis: Process, Principles, and Techniques*. Wiley India Private Limited, 2008.

[3] A. Bertolino. Software testing research: Achievements, challenges, dreams. In *Future of Software Engineering (FOSE '07)*. IEEE, 2007.

[4] V. Casey. *Software Testing and Global Industry: Future Paradigms*. Cambridge Scholars Publisher, 2008.

[5] M. Ould. *Testing in Software Development*. Cambridge University Press, 1987.

[6] A. Srikanth, K. Nandakishore, N. Venkat, S. Puneet, and S. Praveen Ranjan. Test case optimization using artificial bee colony algorithm. In *Advances in Computing and Communications*. Springer Berlin Heidelberg, 2011.

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

[8] M. Harman. Making the case for morto: Multi objective regression test optimization. In *The 4th International Conference on Software Testing, Verification and Validation Workshops (ICSTW'11)*. IEEE, 2011.

[9] B. Baudry, F. Fleurey, J. Jezequel, and Y. Le Traon. Genes and bacteria for automatic test cases optimization in the .net environment. In *The 13th International Symposium on Software Reliability Engineering (ISSRE'02)*. ACM, 2002.

[10] D. Di Nardo, N. Alshahwan, L. Briand, and Y. Labiche. Coverage-based regression test case selection, minimization and prioritization: A case study on an industrial system. *Software Testing, Verification and Reliability*, 25(4):371–396, 2015.

[11] S. Elbaum, G. Rothermel, and J. Penix. Techniques for improving regression testing in continuous integration development environments. In *The 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE-22)*. ACM, 2014.

[12] D. Berthier. *Pattern-Based Constraint Satisfaction and Logic Puzzles (Second Edition)*. Lulu Publishing, 2015.

[13] M. Khan and F. Khan. Importance of software testing in software development life cycle. *International Journal of Computer Science Issues (IJCSI)*, 11(2), 2014.

[14] E. Alégroth, R. Feldt, and P. Kolström. Maintenance of automated test suites in industry: An empirical study on visual gui testing. *Information and Software Technology*, 73:66 – 80, 2016.

[15] "ISO/IEC/IEEE" international standard - software and systems engineering –software testing –part 1:concepts and definitions. *"ISO/IEC/IEEE" 29119-1:2013(E)*, pages 1–64, 2013.

[16] D. Mosley and B. Posey. *Just Enough Software Test Automation*. Just enough series. Prentice Hall PTR, 2002.

[17] G. Yu-Hsin Chen and P. Wang. Test case prioritization in a specification-based testing environment. *Journal of Software (JSW)*, 9:2056–2064, 2014.

[18] G. Myers. *The Art of Software Testing*. A Wiley-Interscience publication. Wiley, 1979.

[19] A. Basu. *Software quality assurance, testing and metrics*. Prentice hall of India private limited, 2015.

[20] T. Ostrand, E. Weyuker, and R. Bell. Where the bugs are. In *The ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'04)*. ACM, 2004.

[21] J. Moré, B. Garbow, and K. Hillstrom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software*, 7(1):17–41, 1981.

[22] J. Pintér. *Global Optimization: Software, Test Problems, and Applications*, pages 515–569. Springer, 2002.

[23] G. Fraser and F. Wotawa. Redundancy based test-suite reduction. In *Fundamental Approaches to Software Engineering*, pages 291–305. Springer Berlin Heidelberg, 2007.

[24] K. Alagarsamy. A synthesized overview of test case optimization techniques. *Journal of Recent Research in Engineering and Technology*, 1(2):1–10, 2014.

[25] S. Biswas, M. Kaiser, and S. Mamun. Applying ant colony optimization in software testing to generate prioritized optimal path and test data. In *The International Conference on Electrical Engineering and Information Communication Technology (ICEEICT'15)*. IEEE, 2015.

[26] A. Windisch, S. Wappler, and J. Wegener. Applying particle swarm optimization to software testing. In *The 9th Annual Conference on Genetic and Evolutionary Computation (GECCO '07)*. ACM, 2007.

[27] S. Dahiya, J. Chhabra, and S. Kumar. Application of artificial bee colony algorithm to software testing. In *The 21st Australian Software Engineering Conference (ASWEC'10)*. ACM, 2010.

[28] P. Srivastava. Optimization of software testing using genetic algorithm. In *The Information Systems, Technology and Management (ISM'09)*. Springer, 2009.

[29] G. Rothermel, R. Untch, and C. Chengyun. Prioritizing test cases for regression testing. *IEEE Transactions on Software Engineering*, 27(10):929–948, 2001.

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

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

[32] S. Tahvili, W. Afzal, M. Saadatmand, M Bohlin, D. Sundmark, and S. Larsson. Towards earlier fault detection by value-driven prioritization of test cases using fuzzy topsis. In *The 13th International Conference on Information Technology : New Generations (ITNG'16)*. ACM, 2016.

[33] S. Tahvili, M. Saadatmand, and M. Bohlin. Multi-criteria test case prioritization using fuzzy analytic hierarchy process. In *The 10th International Conference on Software Engineering Advances (ICSEA'15)*. IARIA, 2015.

[34] A. Rikalovic, I. Cosic, and D. Lazarevic. Gis based multi-criteria analysis for industrial site selection. *Procedia Engineering*, 69:1054 – 1063, 2014.

[35] S. Elbaum, A. Malishevsky, and G. Rothermel. Test case prioritization: A family of empirical studies. *IEEE Transactions on Software Engineering*, 28(2):159–182, 2002.

[36] R. Abid and A. Nadeem. A novel approach to multiple criteria based test case prioritization. In *The 13th International Conference on Emerging Technologies (ICET'17)*. IEEE, 2017.

[37] K. Wang, T. Wang, and X. Su. Test case selection using multi-criteria optimization for effective fault localization. *Computing*, 100(8):787–808, 2018.

[38] J. Jones and M. Harrold. Test-suite reduction and prioritization for modified condition/decision coverage. *IEEE Transactions on Software Engineering*, 29(3):195–209, 2003.

[39] S. Arlt, T. Morciniec, A. Podelski, and S. Wagner. If a fails, can b still succeed? inferring dependencies between test results in automotive system testing. In *The 8th International Conference on Software Testing, Verification and Validation (ICST'15)*. IEEE, 2015.

[40] P. Caliebe, T. Herpel, and R. German. Dependency-based test case selection and prioritization in embedded systems. In *The 5th International Conference on Software Testing, Verification and Validation (ICST'12)*. IEEE, 2012.

[41] M. Broy. Challenges in Automotive Software Engineering. In *The 28th International Conference on Software Engineering (ICSE'06)*. IEEE, 2006.

[42] S. Tahvili, L. Hatvani, M. Felderer, W. Afzal, M. Saadatmand, and M. Bohlin. Cluster-based test scheduling strategies using semantic relationships between test specifications. In *The 5th International Workshop on Requirements Engineering and Testing (RET'18)*, 2018.

[43] N. Iqbal, W. Rizwan, and J. Qureshi. Improvement of key problems of software testing in quality assurance. *Computing Research Repository (CoRR)*, 1202.2506, 2012.

[44] S. Naik and P. Tripathy. *Software Testing and Quality Assurance: Theory and Practice*. Wiley, 2008.

[45] D. Harter, K. Mayuram, and S. Slaughter. Effects of process maturity on quality, cycle time, and effort in software product development. *Management Science*, 46(4):451–466, 2000.

[46] M. Harman. Making the case for morto: Multi objective regression test optimization. In *The 4th International Conference on Software Testing, Verification and Validation Workshops (ICSTW'11)*. IEEE, 2011.

[47] P. McMinn. Search-based software test data generation: A survey: Research articles. *Software Testing, Verification and Reliability*, 14(2):105–156, 2004.

[48] S. Tahvili. *A Decision Support System for Integration Test Selection*. 2016. Licentiate Thesis Dissertation, Mälardalen University, Sweden.

[49] L. Biacino and G. Giangiacomo. Fuzzy logic, continuity and effectiveness. *Archive for Mathematical Logic*, 41(7):643–667, 2002.

[50] S. Ulewicz and B. Vogel-Heuser. System regression test prioritization in factory automation: Relating functional system tests to the tested code using field data. In *The 42nd Annual Conference of the IEEE Industrial Electronics Society (IECON'16)*. IEEE, 2016.

[51] S. Vöst and S. Wagner. Trace-based test selection to support continuous integration in the automotive industry. In *The International Workshop on Continuous Software Evolution and Delivery (CSED'16)*. IEEE, 2016.

[52] M. Parsa, A. Ashraf, D. Truscan, and I. Porres. On optimization of test parallelization with constraints. In *The 1st Workshop on Continuous Software Engineering co-located with Software Engineering (CSE'16)*. CEUR-WS, 2016.

[53] I. Burnstein, T. Suwanassart, and R. Carlson. Developing a testing maturity model for software test process evaluation and improvement. In *The International Test Conference on Test and Design Validity (ITC'96)*. IEEE, 1996.

[54] M. Usaola and P. Mateo. Mutation testing cost reduction techniques: A survey. *IEEE Software*, 27(3):80–86, 2010.

[55] S. Slaughter, D. Harter, and K. Mayuram. Evaluating the cost of software quality. *Communications of the ACM*, 41(8):67–73, 1998.

[56] B. Boehm and V. Basili. Software defect reduction top 10 list. *The Computer Journal*, 34(1):135–137, 2001.

[57] S. Tahvili, M. Saadatmand, S. Larsson, W. Afzal, M. Bohlin, and D. Sundmark. Dynamic integration test selection based on test case dependencies. In *The 11th Workshop on Testing: Academia-Industry Collaboration, Practice and Research Techniques (TAICPART'16)*. IEEE, 2016.

[58] T. Gorschek, P. Garre, S. Larsson, and C. Wohlin. A model for technology transfer in practice. *IEEE Software*, 23(6):88–95, 2006.

[59] S. Fleeger and W. Menezes. Marketing technology to software practitioners. *IEEE Software*, 17(1):27–33, 2000.

[60] V. Basili, F. E. McGarry, R. Pajerski, and M. Zelkowitz. Lessons learned from 25 years of process improvement: the rise and fall of the nasa software engineering laboratory. In *The 24th International Conference on Software Engineering (ICSE'02)*. ACM, 2002.

[61] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131, 2008.

[62] Testomat project- the next level of test automation. http://www.testomatproject.eu.

[63] W. Afzal, H. Bruneliere, W. Di Ruscio, A. Sadovykh, S. Mazzini, E. Cariou, D. Truscan, J. Cabot, A. Gómez, J. Gorroñogoitia, L. Pomante, and P. Smrz. The megam@rt2 ecsel project: Megamodelling at runtime – scalable model-based framework for continuous development and runtime validation of complex systems. *Microprocessors and Microsystems*, 61:86 – 95, 2018.

[64] Imprint-innovative model-based product integration testing. http://www.sics.se/projects/imprint.

[65] Tocsyc - testing of critical system characteristics. https://www.sics.se/projects/tocsyc2.

[66] Electric multiple unit class 490 – hamburg, germany. https://www.bombardier.com/en/transportation/projects/project.ET-490-Hamburg-Germany.html.

[67] Bombardier wins order to supply new generation movia metro fleet for stockholm. http://ir.bombardier.com/en/press-releases/press-releases/44772-bombardier-wins-order-to-supply-new-generation-movia-metro-fleet-for-stockholm.