Mälardalen University Licentiate Thesis 328

Model-Based System Testing of Safety-Critical Embedded Software

Muhammad Nouman Zafar

Mälardalen University

Mälardalen University Press Licentiate Theses
No. 328

MODEL-BASED SYSTEM TESTING OF SAFETY-CRITICAL EMBEDDED SOFTWARE

Muhammad Nouman Zafar

2022

School of Innovation, Design and Engineering

ISBN 978-91-7485-560-9
ISSN 1651-9256
Printed by E-Print AB, Stockholm, Sweden

Abstract

System-level testing of safety-critical embedded systems is complex and costly. MBT has shown promising results in terms of fault detection effectiveness and efficiency of test generation and execution. However, the industrial adoption of MBT approaches is slow and limited to specific industries and domains. Moreover, the strengths and weaknesses of MBT in industrial settings need to be thoroughly evaluated to find an optimal testing strategy.

The objective of this thesis is to adapt, compare, and evaluate the effectiveness and efficiency of MBT to help industrial practitioners in the testing of safety-critical embedded software. We have divided this objective into three subgoals. To achieve the first subgoal, we have explored multiple state-of-the-art MBT tools and evaluated the selected tool, GraphWalker (GW), in terms of modeling notations, generation algorithm, stopping conditions, and model completeness. To achieve the second subgoal, we have proposed a Model-Based Test scrIpt GenEration fRamework (TIGER), based on GW, to generate system-level test artifacts (i.e., test cases and test scripts). Based on the proposed framework, we implemented two test script generation tools for combinatorial testing (CT) and MBT. Finally, to achieve the last subgoal, we performed a comparative analysis between test suites developed using MBT, CT, and manual industrial practices.

The results showed that the MBT-generated test suites using the edge coverage criterion tend to cover each requirement multiple times while achieving the same level of requirement coverage as the manually written test suites. Moreover, MBT provided higher Modified Decision and Condition Coverage (MC/DC) than CT and manual testing. On the other hand, CT came out as the most efficient technique in terms of the time required to generate and execute tests as well as achieving the highest fault detection rate with 3-ways and 4-ways interaction strength. Hence, based on the results, we conclude that manual industrial testing will benefit from MBT and CT for improved coverage and fault detection.

Sammanfattning

Testning på systemnivå av säkerhetskritiska inbyggda system är komplext och kostsamt. MBT har visat lovande resultat när det gäller feldetekteringseffektivitet och effektivitet i testgenerering och testutförande. Den industriella utvecklingen av MBT-metoder är dock långsam och begränsad till specifika branscher och domäner. Dessutom måste styrkorna och svagheterna hos MBT i industriella miljöer utvärderas noggrant för att hitta en optimal teststrategi.

Syftet med denna avhandling är att anpassa, jämföra och utvärdera effektiviteten hos MBT för att hjälpa industriutövare att testa säkerhetskritisk inbyggd programvara. Vi har delat in detta mål i tre delmål. För att uppnå det första delmålet har vi utforskat flera av de främsta MBT-verktygen och utvärderat det valda verktyget, GraphWalker (GW), i termer av modelleringsnotationer, genereringsalgoritm, stoppförhållanden och modellfullständighet. För att uppnå det andra delmålet har vi föreslagit ett Model-Based Test ScrIpt Generation Framework (TIGER), baserat på GW, för att generera testartefakter på systemnivå (d.v.s. testfall och testskript). Baserat på det föreslagna ramverket implementerade vi två testskriptgenereringsverktyg för kombinatorisk testning (CT) och MBT. Slutligen, för att uppnå det sista delmålet, utförde vi en jämförande analys mellan testsviter utvecklade med MBT, CT och manuell industriell praxis.

Resultaten visade att de MBT-genererade testsviterna som använder kanttäckningskriteriet tenderar att täcka varje krav flera gånger samtidigt som de uppnår samma nivå av kravtäckning som de manuellt skrivna testsviterna. Dessutom gav MBT högre Modified Decision and Condition Coverage (MC/DC) än CT och manuell testning. Å andra sidan kom CT ut som den mest effektiva tekniken när det gäller den tid som krävs för att generera och utföra tester samt att uppnå den högsta feldetekteringshastigheten med 3-vägs och 4-vägs interaktionsstyrka. Baserat på resultaten drar vi därför slutsatsen att manuell industriell testning kommer att dra nytta av MBT och CT för förbättrad täckning och feldetektering.

To my parents

Acknowledgments

I am grateful to my supervisors, Wasif Afzal and Eduard Paul Enoiu, for their invaluable guidance and ideas in designing the research activities, providing constructive feedback, and encouragement throughout the thesis. I would also like to thank them for providing me their support to resolve those "hard to address" comments from reviewers that required many changes. I still have a lot to learn from them. I would also like to thank all my co-authors and collaborators for their contributions and support.

I have been fortunate enough to work on the real industrial problem at Alstom (formerly Bombardier) Transportation AB, Sweden. This was made possible by the support of Ola Sellin, Athanasios Stratis, and Inderjeet Singh. I would also like to thank Göran Sohlman, and all the members of the testing and development team at Alstom for helping and providing me with training sessions to understand the fundamental processes and activities followed at Alstom for the development and testing of software applications.

Many thanks to my academic brothers Muhammad Abbas and Damir Bilic for their support and encouragement. Also, special thanks to all my fellow Ph.D. students at MDU.

I would also like to pay infinite and endless gratitude to my parents, brother, and sister for always believing in me and providing me with the confidence, support, and strength that carried me through so many tough times.

The work presented in this thesis has received funding from the European Union's Horizon 2020 research and innovation program under the project 'Adeptness'.

Muhammad Nouman Zafar, Västerås, October, 2022

List of Publications

Papers included in this thesis[1]

Paper A: Muhammad Nouman Zafar, Wasif Afzal, Eduard Paul Enoiu, Athanasios Stratis, Aitor Arrieta, Goiuria Sagardui. *“Model-Based Testing in Practice: An Industrial Case Study using GraphWalker"*. In the 14th Innovations in Software Engineering Conference 2021 (ISEC 2021).

Paper B: Muhammad Nouman Zafar, Wasif Afzal, Eduard Paul Enoiu, Athanasios Stratis, Ola Sellin. *“A Model-Based Test Script Generation Framework for Embedded Software"*. In the 17th Workshop on Advances in Model Based Testing (A-MOST 2021).

Paper C: Muhammad Nouman Zafar, Wasif Afzal, Eduard Paul Enoiu. *“Evaluating System-Level Test Generation for Industrial Software: A Comparison between Manual, Combinatorial and Model-Based Testing"*. In the 3rd ACM/IEEE International Conference on Automation of Software Test 2022 (AST 2022).

Paper D: Muhammad Nouman Zafar, Wasif Afzal, Eduard Paul Enoiu. *“An Empirical Evaluation of System-Level Test Effectiveness for Safety-Critical Software"*. Submitted to the 22nd IEEE International Conference on Software Quality, Reliability, and Security 2022 (QRS 2022).

[1]The included papers have been reformatted to comply with the thesis layout.

Publications, not included in this thesis

Paper W: Muhammad Nouman Zafar, Wasif Afzal, Eduard Paul Enoiu. "*Towards a Workflow for Model-Based Testing of Embedded Systems*". Published in The 12th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation (A-TEST 2021).

Paper X: Asia Shahab, Ayesha Naseer, Muhammad Nouman Zafar, Aamer Nadeem. "*Detection of Energy Bugs in Android Applications: A Systematic Literature Review*". Published in 18th International Conference on Frontier of Information Technology 2021 (FIT 2021).

Paper Y: Iqra Qasim, Muhammad Waseem Anwar, Farooque Azam, Hanny Tufail, Wasi Haider, Muhammad Nouman Zafar. "*A Model-Driven Mobile HMI Framework (MMHF) for Industrial Control Systems*". Published in IEEE Access, vol. 8, Article ID10827, 2020 (ACCESS'20).

Paper Z: Hanny Tufail, Farooque Azam, Muhammad Waseem Anwar, Muhammad Nouman Zafar, Abdul Wahab Muzaffar, Wasi Haider. "*A Model-driven Alarms Framework (MAF) with Mobile Clients Support for Wide-ranging Industrial Control Systems*". Published in IEEE Access, vol. 8, pp.174279–174304, 2020 (ACCESS'20).

Contents

I

Thesis

Chapter 1

Introduction

The testing of a safety-critical embedded software requires a thorough analysis of the system's correctness, which is a costly and complex process [1]. It is carried out at different levels during the development of a system (i.e., unit, integration, and system) to guarantee its reliability and availability along with mitigation of risk factors that can cause catastrophic events due to abrupt behavior of such systems [2]. Each level of testing deals with a different number of input/output space, communication formats, and test objectives. At the unit and integration level of testing, the input/output space is a lot smaller and the test objectives for the validation process are limited to the functionality of a module or the interaction between different modules, respectively [3]. On the other hand, the testing complexity of a system grows with the scope of a system due to bigger input/output space and end-to-end functionality of integrated modules at the system level. System testing deals with the validation of a whole system or a subsystem to ensure that all the components and modules are working according to functional and non-functional behavior specified in the requirement document. For a safety-critical system, it also includes the testing of a system using risk analysis reports based on interactions between different software components [4]. As the complexity, size, and scope of a safety-critical system increase, the cost and effort of testing also increase, which consequently affects the overall cost of the development process. Hence, there is a need to research into cost-efficient testing techniques while ensuring the quality of safety-critical systems.

There are various automated testing tools and techniques (e.g., combinatorial testing (CT), random testing, search-based testing) that have been used to generate high-quality test cases while reducing the cost of a testing process [5]. In this regard, Model-based Testing (MBT) techniques have also shown promising results [6]. Different studies (e.g., [7], [8], [9], [10]) have shown multiple benefits of model-based approaches for the testing of embedded systems. These benefits include the aid provided by modeling of a SUT to verify and validate system requirements at the initial level of the development process, generation of test artifacts before the development of a real system, and increasing the test coverage as well as fault detection rate while reducing the cost of the testing process in terms of time, effort, and resources [11].

Motivation: In literature, multiple state-of-the-art MBT approaches (e.g., [12], [13], [14], [15]) exist for the testing of safety-critical embedded systems. However, industrial adoption of MBT proposed solutions and approaches is slow due to platform-dependent and domain-centric solutions focusing on a specific scope of a system [6] [16]. The lack of industrial success stories and practical guidelines, as well as limited empirical and evidence-based studies are also among the reasons for the limited adoption of MBT techniques [17]. Furthermore, we have found few studies (i.e. [18] and [19]) that have reported the details on the concretization process (executable test cases) at system and integration levels. This motivates us to empirically investigate the use of MBT techniques in practice for the testing of safety-critical embedded systems at the system level.

Summary of the Contributions: In this thesis, we have examined the efforts of modeling a safety-critical system in terms of completeness and representativeness using an open-source MBT tool (i.e. GraphWalker (GW)). We further evaluated the behavioral differences between the abstract test cases generated by modelling notations, generation algorithms and stopping conditions of the selected tool and manually written test cases using an industrial case study. We have also proposed a Model-Based Test scrIpt GenEration fRamework (TIGER) for the testing of safety-critical embedded systems and defined mapping rules for the concretization of GW-generated abstract test cases along with tool support to generate test artifacts (i.e. test cases and test scripts) at the system level and evaluated the proposed approach in close industrial collaboration. We have developed a cost model to measure the efficiency of a testing technique and used Modified Condition Decision Coverage (MC/DC) and requirement coverage to examine the effectiveness. Requirement coverage is an important criterion for our industrial partner at system level. Multiple safety standards such as ISO 26262 [20], IEC 61508 [21], EN 50128, and EN 50657 [22] also recommend the MC/DC coverage for the generation and evaluation of test suites [23]. Multiple studies (e.g., [24], [25], etc) also showed the effectiveness of MC/DC adequate test suite for the testing of safety-critical systems. Similarly, different studies (e.g., [26], [27], [28], [29], etc.) showed the effectiveness of CT-generated test suite as well as used it to achieve MC/DC adequacy for the testing of safety-critical systems based on parameters/signals interactions at the system level. Hence, we have also performed a comparative analysis between the test suites generated by the MBT framework (TIGER), CT, and written manually by industrial practitioners, in terms of MC/DC, requirement coverage, and efficiency (i.e. test generation and execution). Furthermore, it is also evident from the literature (e.g., [30], [31], [32], [33], etc.) that the behavioral and structural coverage alone is not sufficient to evaluate the quality of a test suite. Hence, to provide a complete evaluation of test suites, we have also examined the fault detection capabilities of each test suite using mutation analysis.

Results: The comprehensive evaluation of the testing techniques used in this thesis shows that:

- The proposed framework (i.e., TIGER) is capable of generating system-level test cases and scripts to validate a safety-critical SUT.

- Test suite generated by MBT using a random generator algorithm and edge coverage criterion provided higher MC/DC than combinatorial and manual testing, and tends to cover each requirement multiple times while achieving the same level of requirement coverage.

- MBT-generated test suite acted as a superset that contained the highest number of similar and unique test cases across the test suites generated by other selected techniques (i.e. CT and manual testing).

- MBT-generated test suites are more relevant to manual specification-based testing in terms of structure and completeness (i.e, input, expected output and constraints) and will complement manual testing in practice to achieve higher MC/DC coverage.

- CT was found to be the most efficient technique in terms of test generation and execution time with an additional cost of writing the test scripts along with the inclusion of expected output and timing constraints manually in the test suite.
- MBT-generated test suite achieved a higher fault detection rate than manual testing practices in the industry. However, regardless of the highest MC/DC, it provided a slightly lower fault detection rate than the CT-generated test suite with high combinational interaction strength (i.e. 3-ways and 4-ways).

Outline of the Thesis: The thesis consist of two parts. Part I provides an overview of the conducted research and organized as follows: Chapter 2 provides a brief overview of the background along with related work, Chapter 3 presents the research goals, methodology and research process, Chapter 4 summarizes the included papers and contributions, Chapter 5 deals with discussion and limitations of the thesis followed by conclusion and future work in Chapter 6. Part II includes the published papers, which have been formatted to comply with the format of the thesis.

Chapter 2

Background & Related Work

This chapter explains the fundamentals of the development of safety-critical embedded software, the purpose of its testing in industry, system-level testing techniques, test coverage, and mutation analysis followed by related work that has been carried out in similar domains.

2.1 Embedded Safety-Critical Software and its Testing

Embedded systems are composed of hardware and software components that interact with the real world through sensors and actuators to manipulate the environment [34]. The hardware of the embedded systems is controlled by a software system that is responsible for regular and safety-critical functions according to requirement specifications. These systems are specially designed and developed to facilitate humans to execute complex tasks with a reduction of effort and time. Such systems have widely been adopted in different domains such as aviation, transportation, nuclear power plants, etc and failure of these systems can cause harm to human life, and the environment as well as can cause economic problems [35]. Hence, the development of the software controlling the safety-critical function requires a certain degree of software verification and validation through a defined process according to certain safety standards and regulations to ensure its reliability.

2.1.1 Development of Safety-Critical Software in Industry

The development of safety-critical software in the industry starts with the requirement analysis based on qualitative and quantitative analysis techniques such as fault tree analysis, expert analysis, etc. [36]. After the analysis of the requirements, the safety-critical software is usually implemented for specially designed computers known as Programmable Logic Controllers (PLCs), which are responsible for performing safety-critical operations of a system. PLCs receive the input signals from the sensors, perform computational logic and send the instructions via a computer network as the output to different modules and subsystems to perform the safety-specific tasks. There are several programming languages defined by the International Electrotechnical Commission (IEC) [37] to implement safety-critical software applications for PLCs (e.g., Instruction List (IL), Structured Text (ST), Functional Block Diagram (FBD), etc.). In this context, FBD is a renowned language and has been adopted by Alstom to develop the system under consideration.

FBD uses a graphical modeling notation representing the different functions and functional blocks such as arithmetic, selection, comparison, etc. These function blocks are connected to each other through input and output variables to define the functional properties and relationships between different components of the software application according to defined functional and non-functional requirements. Figure 2.1 represents an example of an FBD program consisting of the arithmetic operators (AND, OR, NOT, XOR), a latch (SR), and a timer (TON) function block. It takes six parameters/signals as inputs and provides one output based on the logic represented by functional blocks and after a delay of five seconds. The developed FBD programs are then complied through specific industrial compiling tools and transformed into the source and machine code.

2.1.2 Testing of Safety-Critical Software in Industry

Software verification and validation of safety-critical software is an iterative process that is carried out throughout the development life cycle to ensure its behavioral functionality according to system requirements. In industrial practices, the testing process starts in parallel to the software development process following the V-model which has been widely accepted by the industrial prac-

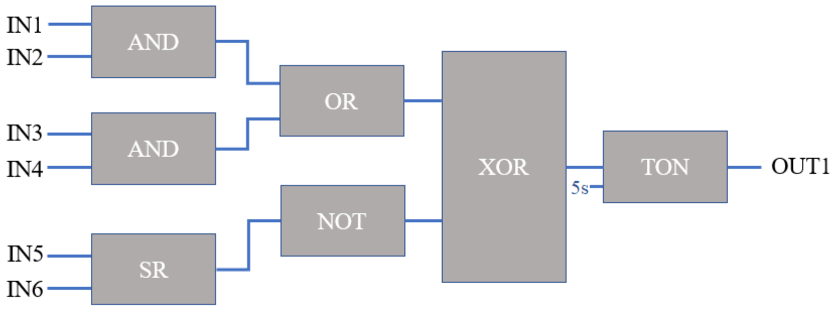

Figure 2.1: An example of FBD program with six inputs and one output

titioners developing the embedded software [38] [39] [40]. However, different versions of V-models [41] are followed by different industries according to their business needs. In this section, we have presented a simplified version of the V-model as shown in Figure 2.2 to provide the fundamental concepts used in this thesis.

The V-Model is divided into four phases i.e., the development phase, test design phase, testing phase, and implementation phase. On the left side of Figure 2.2, the development phase is illustrated in which each activity of the development life cycle is presented at the abstract level [40]. The details in the functional specification about the system increase at each level correspondingly. Level 0 deals with the functional specification from a customer perspective, level 1 provides the functional design of the complete system, level 2 specifies the detailed description of data transfer and communication between each module and external environment whereas level 3 deals with the detailed functional specification of each module at the unit level. The test design phase is carried out in parallel to the development phase in which test specifications for each level are created accordingly. The implementation phase deals with the actual development of each module of the system to realize the system requirements. After the development of each module, the test designs are executed in an iterative and incremental manner during the testing phase which has been illustrated on the right side of the figure.

The test execution phase in the embedded system industry is usually carried out at three levels (i.e., Model-in-the-loop (MiL), Software-in-the-Loop (SiL), and Hardware-in-the-Loop (HiL)) [34]. At the MiL level, the tests are

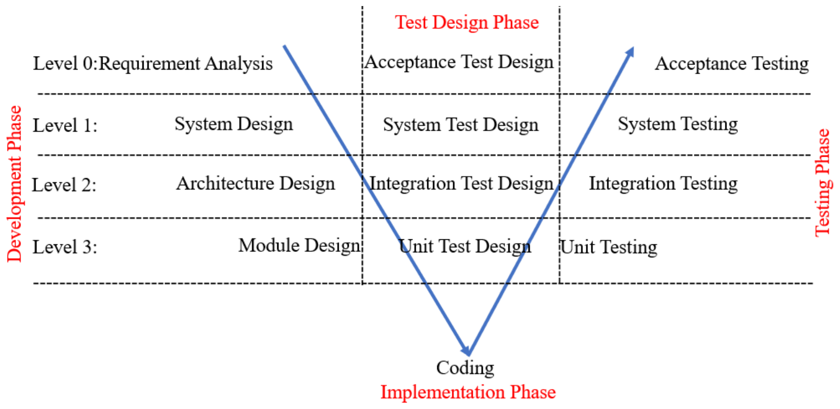

Figure 2.2: A simplified version of a V-model

executed on the model representing the requirements of the system to verify the conformance of the model with requirements as well as computational logic. Whereas, at the SiL level, the tests are executed on the actual software and using experimental hardware (i.e. simulation depicting the behaviors of actual hardware). On the other hand, at the HiL level, the tests are executed on the actual software and hardware in a simulated or virtual test environment.

The testing complexity and effort needed for each activity in the testing phase grows with each step [39]. Each defined activity contains different test objectives, input/output space, and communication format. At the unit and integration level, the test objectives, as well as the input/output space, are limited to the functionality of smaller parts of the system. Whereas the complexity at the system level increases with the increase of the input/output space and the number of integrated modules which consequently increases the effort of testing at the system level.

2.2 System-Level Testing Techniques

The primary purpose of the software testing technique is to develop high-quality test artifacts (i.e., test cases and test scripts) to validate the system requirements and its reliability. There exist different automated testing techniques such as search-based testing [42], random testing [43], model-based testing (MBT) [44], and combinatorial testing (CT) [45] which can be used to generate test artifacts to validate a SUT automatically with the reduction of cost in terms of effort and time. However, manual testing is still considered an equally prevalent technique and has been widely applied in the industry as a complement to automated software testing [46]. These techniques can be applied at different levels of testing (e.g., component, integration testing). However, in this thesis, CT, MBT, and manual testing are conducted at the system level to empirically evaluate the system's compliance with its specified requirements. There exist multiple studies (e.g., [47], [48], [26], [49], etc.) in the literature which have shown the advantages of CT and MBT in terms of effectiveness and efficiency to validate safety-critical software at the system level.

2.2.1 Manual Testing

In manual testing, the test cases for a safety-critical system are designed manually following certain safety standards defined by different organizations (e.g., ISO [20], IEC [21], etc.). It utilizes requirement and test specifications to design the test cases based on test objective and different structural or behavioral coverage criteria such as statement coverage and input space partitioning, respectively. The test cases are written in natural language in the form of test steps consisting of input, expected output, and constraints according to system requirements. These test cases are then converted into concrete test cases or test scripts which can be executed manually or automatically on the SUT to produce test verdicts.

2.2.2 Combinatorial Testing

CT is an automated testing technique that is used to generate a test suite in form of a coverage array based on the interaction possibilities between the input parameters of the system [50]. Each row in generated covering array represents a test case that is used to validate functional requirements. It provides a t-ways testing strategy to generate a relevant, unique, and a finite number of test cases in a test suite using different meta-heuristic algorithms. The number of test cases generated by CT depends on the value of *t*, which is also known as interaction strength. For example, if a test suite is generated using a pairwise (i.e., 2-ways) interaction strength, the test cases cover all combinations between the pairs of inputs parameters at least once.

2.2.3 Model-based Testing

MBT [51] is one of the advanced automated techniques that can be used to generate test cases as well as test scripts automatically using an explicit model representing the behavioral and environmental aspects of the system. The model used by the MBT can be created using different modeling notations such as Unified Modelling Language (UML), system sequence diagram (SSD), Finite State Machine (FSM), etc. depending on the test objectives. For example, for system-level testing, an FSM model consisting of nodes (i.e., depicting the states of the system), edges (i.e., illustrating the transition between these states), and guard conditions (i.e., Boolean constraints representing the expected behavior of the system) is created. Once a model is available, it is then used to generate abstract test cases based on generation algorithms (e.g., random, weight random) and stopping conditions (e.g., edge coverage, requirement coverage). These abstract test cases are then transformed into concrete test cases/test scripts and executed on the SUT to generate test verdicts.

2.3 Test Coverage

The test coverage in software testing is a metric that evaluates the thoroughness of a test suite by measuring the degree of covered elements (e.g., code and requirements) provided by a test suite at the design or implementation level [52]. There exist different test coverage criteria such as branch coverage, statement coverage, and requirement coverage to evaluate and generate a test suite. However, in this thesis, we have performed an empirical comparative evaluation of the test suite generated by the above-mentioned techniques using requirement coverage and Modified Condition Decision Coverage (MC/DC). We have selected the requirement coverage criterion as it is a de facto standard for our industrial partner (i.e., Alstom Transportation AB) for the creation of test suites. Whereas MC/DC has been widely accepted by industrial practitioners as well as different safety standards (e.g., ISO 26262 [20], IEEE 50657 [22]) recommend it for the testing of safety-critical systems. Moreover, different studies (e.g., [24], [53]) showed evidence of the effectiveness of MC/DC adequate test suite for the validation of safety-critical systems.

2.3.1 Requirement Coverage

For a safety-critical system, it is important to analyze that each safety and domain requirements of the system documented in the requirement specification is covered by a test suite at least once [51]. Requirement coverage is one of the basic black-box coverage criteria that is used to access the behavioral coverage of a test suite. It measures the total number of implemented requirements, determines the undocumented requirements in the implementation as well as the total number of test cases required to cover each requirement.

2.3.2 Modified Condition Decision Coverage

MC/DC is one of the structural coverage criteria to measure the code coverage of the implemented functionalities. It is also known as cost-effective and a stronger coverage criterion by providing relatively higher logical predicates and path coverage of implemented code than decision and condition/decision coverage [54]. It evaluates a test suite based on the number of invoked entry and exit points in a program, the number of possible outcomes covered for each condition, the number of possible outcomes covered for each decision, and the number of covered conditions having an independent effect on the decision [55].

2.4 Mutation Analysis

The test coverage criteria can be used to measure the area of code covered by a test suite and to provide information about the possibilities to improve test adequacy. However, different studies (e.g., [30], [31]) showed that, regardless of high test coverage, multiple factors can affect the fault detection effectiveness of a test suite. Hence, in this thesis, to provide a complete evaluation, we have also measured the fault detection effectiveness of generated test suites using mutation analysis.

Mutation analysis [56] is an evaluation technique that is used to determine the fault detection rate of a test suite. It involves creating different versions of the original program and inducing small faults representing common programming errors, syntactical mistakes, or logical errors in each version, systematically. These faulty versions of the original program are known as mutants. The mutant can be categorized as equivalent and non-equivalent mutants. A mutant is said to be equivalent if it exhibits similar behavior to the original program that can not be killed by any test case in a test suite. Whereas a mutant exhibiting different behavior than the original program is considered a non-equivalent mutant that can be killed by potential test cases in a test suite. After the creation of mutants, the test suites are designed by a particular testing technique and executed on the original and mutated versions of the program. If there is a contradiction between the test results of executed test suites on the original program and the mutant version, then the mutant is said to be killed otherwise it is alive. The mutation score is then calculated based on strong mutation (output-only oracle) and weak mutation (internal state change or internal oracle) by determining the number of killed and alive mutants.

2.5 Related Work

This section deals with the state-of-the-art studies that have proposed, investigated, and compared model-based testing techniques to generate test artifacts (i.e. test cases and test scripts) in different domains. There exist various studies (e.g. [9], [8], [12], [57], [58], [59], [60], [53], [61], [62], [10], [63], [17], [64]) that have proposed multiple model-based approaches and tools as well as evaluated the proposed approaches in terms of test generation efficiency and fault detection effectiveness. A summary of this literature is presented below.

MBT proposed approaches: Model-Based Testing (MBT) showed promising results while dealing with the productivity and time-to-market constraints of embedded systems. It not only promises to improve reliability and reusability features but also expedites the design verification process [57]. Guan and Offutt [9] proposed a novel MBT approach for the testing of real-time properties in component-based embedded systems. Particularly, the authors introduced the generation of a Component-based Real-time Embedded Model-based Test Graph (CREMTEG) from sequence and state diagrams. Subsequently, test cases are generated using pre-defined test criteria. The validation is performed through prototype implementation in industrial settings. In another study, Marinescu et al. [8] proposed MBT approach by extending EAST-ADL models with timed automata semantics to generate concrete test cases in python to perform real-time testing. Enoiu et al. [12] [59] proposed a tool-supported approach using model checking to generate test cases and provided an empirical evaluation of the proposed approach by applying it for the validation of safety-critical embedded systems at unit level. The results showed that the proposed approach positively affected the testing process in terms of test coverage and the tool was found to be efficient for test generation. Tseng et al. [60] introduced an ontology-based MBT approach where sequence diagrams are generated from textual Safety Analysis Report (SAR). The proposed approach is also used to generate scenario test cases for black-box testing. Gannous et al. [58] introduced the integration of safety certification in MBT with the proposal of a Model-Combinatorial based testing (MCbt) framework that allows the generation of both normal behavior and failure paths along with failure mitigation paths. The applicability of the proposed approach is demonstrated through a railroad crossing control system case study. The results showed that the proposed approach is highly efficient in terms of time for testing safety-critical embedded systems.

Comparative studies: El-Fakih et al. [61] provided an empirical evaluation of test suites generated from extended finite state machine (EFSM) using different coverage criteria (i.e., all-uses, edge pair, single transfer faults, transition tour, and prime path with side trip). They carried out the proper statistical evaluation at the model and code level using mutations to determine the fault detection effectiveness of test suites and test coverage. The results showed that the test suite generated using single transfer faults outperformed all the other test suites in terms of fault detection. Moreover, they have found that similar test coverage was achieved by each test suite as well as consistent results were obtained at model and code levels. Christoph et al. [62] performed a comparative evaluation to examine the merits and demerits of MBT and manual testing in terms of effort efficiency and fault detection effectiveness. The result of the evaluation showed MBT as a systematic technique with less testing and analysis effort and provided a higher fault detection rate in detecting functional issues of the system than manual testing. In another study, Stephan et al. [10] provided an evaluation of MBT and manual testing in terms of efficiency as well as reported the lesson learned in adopting the MBT technique for the validation of embedded applications. Jose et al. [63] proposed a hybrid approach based on combinatorial and MBT and performed a comparative evaluation in terms of fault detection effectiveness with manual testing tools. The results illustrated the higher fault detection rate of the proposed approach than manual testing tools.

Experienced reports on MBT: Alegroth et al. [17] conducted interviews with different international MBT experts considering technical and non-technical factors to provide guidance on the adoption and abandonment of MBT practices. The results of the study have drawn multiple implications related to the abstract design of models and the aptitude of an individual for abstraction, collaboration among stakeholders to provide early fault detection, use of scenario-based requirements, etc. In another study, Garousi et al. [64] presented an experience report on the adoption of MBT practices to automate the testing process in the industrial setting of the web application domain. They also reported the lessons learned as well as various benefits of MBT practices including improved test coverage, test design, and fault detection effectiveness.

Chapter 3

Research Overview

This chapter provides a brief description of the research goals along with the methodology used to conduct the research activities to achieve the defined research goals.

3.1 Motivation & Research Goal

A safety-critical system consists of different integrated modules and subsystems communicating with each other to perform real-time operations according to specified requirements [65]. Verification and validation of safety-critical functions is carried out at different levels (i.e., unit, integration, and system level). The scope of testing at the unit and integration level is limited to a component/module implementing a requirement or a limited number of components/modules communicating with each other, respectively. Whereas system-level testing ensures the correctness of a system by developing test cases covering each requirement of the system and determining errors in the implementation violating the system requirements. The severity of testing effort and cost increases at the system level with the increase in the number of integrated components/modules and subsystems. Hence, the focus of this thesis is to adapt, apply and evaluate system-level test generation approaches in an industrial setting.

As mentioned in Section 2.2, different automated testing techniques can be used to generate the system-level test artifacts. MBT is one of the techniques that can be applied throughout the V-model at different integration levels depending on test objectives, modeling language, generation algorithm, test selection criteria and scope of the MBT model [51] [66]. During a development life cycle, MBT is mostly used for the generation of high-level tests focusing on functional and non-functional aspects of a system [51]. At the system level, MBT models represent the SUT as a black box containing the details about the functional and non-functional requirements, and the system environment. At the integration level, MBT models depict the data flow and the expected behaviors between the components interacting with each other. And at the unit level, the MBT model represents the functional aspects of a component of a system. Furthermore, it also addresses two non-trivial problems (i.e., the problem of translating user and consistency requirements into concrete input and the oracle problem) of the validation process of a system by generating test inputs and expected output from the model before the actual implementation of the system [66]. However, the adoption of MBT is slow and limited in the industry due to a lack of industrial success stories and practical guidelines, platform-dependent solutions, and limited empirical studies [17]. This motivates us to empirically investigate MBT in practice to validate safety-critical embedded software at the system level.

Main Objective. To adapt, compare, and evaluate model-based test generation of system-level test artifacts for safety-critical embedded software.

- **Subgoal 1 (SG 1).** To investigate and study state-of-the-art model-based test generation tools, and evaluate the selected tool in terms of model completeness and representativeness in an industrial setting.

- **Subgoal 2 (SG 2).** To develop and apply a system-level model-based test generation approach for safety-critical embedded system in an industrial setting.

- **Subgoal 3 (SG 3).** To perform a comparative analysis between the model-based test generation approach, combinatorial-based, and manual approaches in terms of test coverage, fault detection effectiveness, and efficiency in an industrial setting.

3.2 Research Method

In software engineering, there exist multiple methods (e.g., case study, experiment, survey) to conduct empirical research depending on the research goals, type of analysis, and interpretation of data (i.e. qualitative and quantitative) [67] [68]. In this thesis, we focus on qualitative and quantitative data to provide a better interpretation of research results to the research community and industrial practitioners.

We have used a mixed methodology to conduct our research based on formulated research goals (presented in Section 3.1). We have used case studies and experimentation as research methodologies to evaluate the proposed solution using industrial examples. The mapping of research methods with research goals is shown in Table 3.1.

Table 3.1: Research method mapping with type of data and research goals

Research Goal(s)	Type of Data Analysis	Case Study	Experiment
SG 1	Qualitative	✓	
SG 2	Qualitative & Quantitative	✓	✓
SG 3	Qualitative & Quantitative	✓	✓

3.2.1 Research Process

We have defined our research process in six iterative steps: (1) Review of industrial system and processes, (2) Problem identification and formulation, (3) Proposal of a solution, (4) Solution/Tool implementation, (5) Validation, and (6) Publication of research results. This is captured in Figure 3.1.

1. Review of Industrial System and Processes: We have conducted this step to understand the industrial systems, practices, and processes adopted by industrial practitioners at Alstom. This was done as a preparation to identify their needs and challenges.

2. Problem Identification and Formulation: This step was done to identify industrial problems based on the analysis carried out in the previous

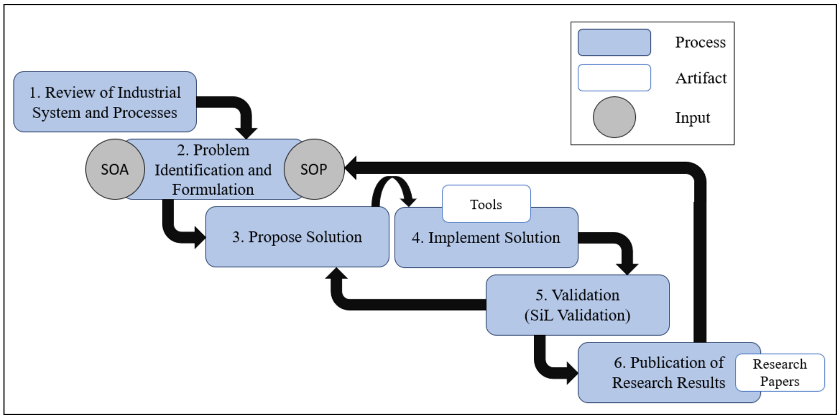
Process
Artifact
Input
1. Review of Industrial System and Processes
2. Problem Identification and Formulation
SOA
SOP
3. Propose Solution
Tools
4. Implement Solution
5. Validation (SiL Validation)
6. Publication of Research Results
Research Papers

Figure 3.1: Overview of the research process applied in the thesis

step. We have also formulated an overall research goal based on the identified problem and challenges that we may encounter while using state-of-the-art MBT tools and techniques for automatic test generation in industrial settings. In Paper A [69], we started with an investigation of MBT tools from the literature. We then evaluated the modeling efforts of the selected tool and behavioral differences between the test cases written manually and generated through MBT.

3. Propose Solution: The investigation based on test generation capabilities and modeling aspects of MBT tools in Paper A led us to propose a model-based test script generation framework to generate test artifacts (i.e., abstract test cases and test scripts) in Paper B [70]. We have also defined some mapping rules to generate concrete test scripts implemented in C# language to utilize the Alstom-specific testing framework in a real industrial setting. The proposed framework was developed and improved through continuous feedback from the testing team at Alstom.

4. Implement Solution: After proposing the MBT framework for test script generation at the system level, an initial prototype (TIGER tool) was developed in C# language using Microsoft Visual Studio and the source code is available on GitHub. Moreover, for evaluation purposes, we have also implemented a combinatorial-based test script generation tool (i.e., CATSGen), which contains similar implementation details and mapping rules to TIGER.

5. Validation: The test scripts generated through the prototypes were deployed and executed using the Alstom testing framework and validated by producing test verdicts in an industrial setting. In Paper C [71] and D, we have conducted a comprehensive and comparative evaluation of the test suites developed by MBT, CT, and manual testing practices. We defined and used different metrics to evaluate and compare the test suites in terms of test coverage, test generation efficiency, and fault detection effectiveness. It is important to mention here that all the validations of generated test scripts and the framework are done at the SiL level using the virtual test environment of a train.

6. Publication of Research Results: The results of investigations and evaluations were published in the form of research papers as shown in Section 4.3.

Chapter 4

Contributions

In this chapter, we present a summary of the contributions to realize the defined goal of this thesis along with a mapping of each contribution towards the subgoals.

4.1 Thesis Contributions

In this thesis, we have proposed an MBT test script generation framework, and empirically investigated, compared, and evaluated the effectiveness and efficiency of it in an industrial environment using multiple case studies and experiments.

(C1) In Paper A [69], we have conducted an investigation for the selection of a MBT tool and have provided an empirical evaluation of the selected tool, GraphWalker (GW), in terms of model completeness and representativeness to generate abstract test cases. Moreover, we have performed a comparative analysis between manually written test cases by industrial practitioners at Alstom in Sweden, and the test cases generated by GW based on a specific modeling notation (FSM), generator algorithms, and coverage criteria in terms of behavioral differences.

(C2) In Paper B [70], we have implemented a Model-Based Test scrIpt GenEration fRamework (TIGER), based on GW, for system-level test scripts in C# language. We have also provided an evaluation of the proposed framework by executing generated test scripts at the SiL level using model-level mutation testing.

(C3) A thorough evaluation of test artifacts (i.e., test cases, test scripts) generated by MBT (using TIGER), combinatorial, and manual testing techniques in terms of MC/DC and requirement coverage is presented in Paper C [71]. We have investigated the MC/DC conditions to determine the least dominant condition affected by each technique. Moreover, we have proposed a cost model to evaluate the efficiency of each technique. Finally, we have performed a comparative analysis to determine the differences and overlaps (i.e., multiple instances of different or similar test inputs, expected outputs, and constraints, respectively) between the generated test cases.

(C4) In Paper D, we have implemented an CT-based test script generation tool (CATSgen) and provided a comparative empirical evaluation of the test suites developed by manual, CT and MBT in terms of fault detection effectiveness. We have also analyzed the sensitivity of each test suite towards mutant operators as well as performed an analysis between the mutation scores and coverage levels achieved by each testing technique.

Figure 4.1 represents a generic, practical workflow of software testing using manual, CT and MBT techniques along with the specific contributions (C). The workflow has six steps: (1) requirements and requirement analysis, (2) domain knowledge of the tester in manual testing and creation of models for MBT and CT, (3) selection of test generation criteria, algorithms, and tools in case of MBT and CT, and the creation of test design in manual testing, (4) creation of test cases and test scripts, (5) execution of test scripts, and (6) generation of test verdict. The arrow from step 5 to step 2 represents the iterative nature of the workflow, which is triggered when a fault occurs during test execution due to a specific reason (e.g., incorrect translation of abstract test cases to concrete test cases/scripts, change/misinterpretation of requirement or parameter/signal).

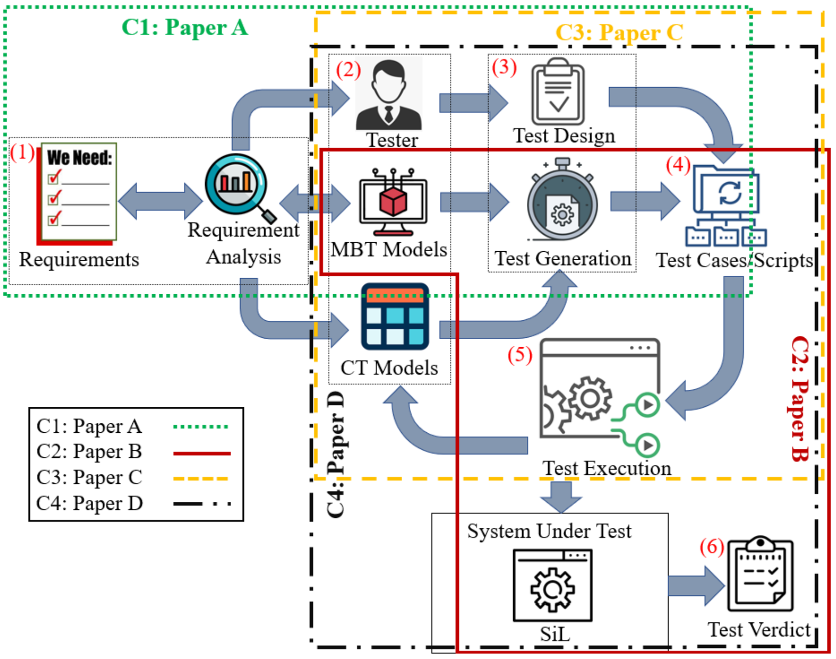

Figure 4.1: A workflow of software testing techniques and mapping of contributed areas of the included papers

4.2 Individual Contribution

I am the primary researcher, driver and author of all the included papers. However, all the other co-authors have contributed with their valuable ideas, discussions and reviews. The supervision team has also contributed in refining the text.

4.3 Included Papers

All the included papers in this thesis have a contribution toward the research goal and the mapping of these contributions is shown in Table 4.1.

In Paper A, we have provided the selection criteria of MBT tools, our initial experience with the selected tool (i.e., GW), and examined the modeling effort

Table 4.1: Mapping of research papers towards the contribution of individual research subgoals

	SG 1	*SG 2*	*SG 3*
C1: Paper A	✓		✓
C2: Paper B		✓	
C3: Paper C			✓
C4: Paper D			✓

in terms of model completeness and representativeness to realize SG 1 whereas the evaluation of MBT and manually created test cases in terms of behavioral differences is done to achieve SG 3. The SG 2 is fulfilled in Paper B in which we have proposed and implemented the model-based Test scrIpt GenEratIon fRamework (TIGER) and applied the proposed framework to generate and validate test scripts in industrial settings. In Paper C and D, we have provided a complete evaluation of the MBT-generated test suite and performed a comparative analysis between MBT, CT, and manually created test suites in terms of test coverage, test effectiveness, and efficiency to accomplish the SG3.

4.3.1 Paper A

Title: Model-Based Testing in Practice: An Industrial Case Study using GraphWalker

Authors: Muhammad Nouman Zafar, Wasif Afzal, Eduard Paul Enoiu, Athanasios Stratis, Aitor Arrieta, Goiuria Sagardui

Status: Published in The 14th Innovations in Software Engineering Conference 2021 (ISEC 2021)

Abstract: Model-based testing (MBT) is a test design technique that supports the automation of software testing processes and generates test artifacts based on a system model representing behavioral aspects of the system under test (SUT). Previous research has shown some positive aspects of MBT such as low-cost test case generation and fault detection effectiveness. However, it is still a challenge for both practitioners and researchers to evaluate MBT tools and techniques in real, industrial settings. Consequently, the empirical evidence regarding the mainstream use, including the modeling and test case

generation using MBT tools, is limited. In this paper, we report the results of a case study on applying GraphWalker, an open-source tool for MBT, on an industrial cyber-physical system (i.e., a Train Control Management System developed by Bombardier Transportation in Sweden), from modeling of real-world requirements and test specifications to test case generation. We evaluate the models of the SUT for completeness and representativeness, compare MBT with manual test cases written by practitioners using multiple attributes as well as share our experiences of selecting and using GraphWalker for industrial application. The results show that a model of the SUT created using both requirements and test specifications provide a better understanding of the SUT from the testers' perspective, making it completer and more representative than the model created based only on the requirements specification alone. The generated model-based test cases are longer in terms of the number of test steps, achieve better edge coverage, and can cover requirements more frequently in different orders while achieving the same level of requirements coverage as manually created test cases.

4.3.2 Paper B

Title: A Model-Based Test Script Generation Framework for Embedded Software

Authors: Muhammad Nouman Zafar, Wasif Afzal, Eduard Paul Enoiu, Athanasios Stratis, Ola Sellin

Status: Published in The 17th Workshop on Advances in Model-Based Testing (A-MOST 2021)

Abstract: The abstract test cases generated through model-based testing (MBT) need to be concretized to make them executable on the software under test (SUT). Multiple researchers proposed different solutions, e.g., by utilizing adapters for the concretization of abstract test cases and the generation of test scripts. In this paper, we propose our Model-Based Test scrIpt GenEration fRamework (TIGER) based on GraphWalker, an open-source MBT tool. The framework is capable of generating test scripts for embedded software controlling functions of a cyber-physical system such as passenger trains developed at Bombardier Transportation AB. The framework follows some defined mapping rules for the concretization of abstract test cases. We have evaluated the generated test scripts using an industrial case study in terms of fault detection.

We have induced faults in the model of the SUT based on three mutation operators to generate faulty test scripts. The aim of generating faulty test scripts is to produce failed test steps and to guarantee the absence of faults in the SUT. Moreover, we have also generated the test scripts using the correct version of the model and executed them to analyze the behavior of the generated test scripts in comparison with manually written test scripts. The results show that the test scripts generated by GW using the proposed framework are executable, provide 100% requirements coverage, and can be used to uncover faults at the software-in-the-loop simulation level of sub-system testing.

4.3.3 Paper C

Title: Evaluating System-Level Test Generation for Industrial Software: A Comparison between Manual, Combinatorial and Model-Based Testing

Authors: Muhammad Nouman Zafar, Wasif Afzal, Eduard Paul Enoiu

Status: Published in the 3rd ACM/IEEE International Conference on Automation of Software Test 2022 (AST 2022)

Abstract: Adequate testing of safety-critical systems is vital to ensure correct functional and non-functional operations. Previous research has shown that testing of such systems requires a lot of effort, thus automated testing techniques have found a certain degree of success. However, automated testing has not replaced the need for manual testing, rather a common industrial practice exhibits a balance between automated and manual testing. In this respect, comparing manual testing with automated testing techniques continues to be an interesting topic to investigate. The need for this investigation is most apparent at system-level testing of industrial systems, where there is a lack of results on how different testing techniques perform concerning both structural and system-level metrics such as Modified Condition/Decision Coverage (MC/DC) and requirement coverage. In addition to the coverage, the cost of these techniques will also determine their efficiency and thus practical viability. In this paper, we have developed cost models for efficiency measurement and performed an experimental evaluation of manual testing, model-based testing (MBT), and combinatorial testing (CT) in terms of MC/DC and requirement coverage. The evaluation is done in an industrial context of a safety-critical system that controls several functions on-board the passenger trains. We have reported the dominant conditions of MC/DC affected by each technique while

generating MC/DC adequate test suites. Moreover, we investigated differences and overlaps of test cases generated by each of the three techniques. The results showed that all test suites achieved 100% requirement coverage except the test suite generated by the pairwise testing strategy. However, MBT-generated test suites were more MC/DC adequate and provided a higher number of both similar and unique test cases. Moreover, unique test cases generated by MBT had an observable effect on MC/DC, which will complement manual testing to increase MC/DC coverage. The least dominant MC/DC condition fulfilled by the generated test cases by all three techniques is the 'independent effect of a condition on the outcomes of a decision'. Lastly, the evaluation also showed CT as the most efficient testing technique amongst the three in terms of time required for test generation and execution, but with an added cost parameter of manual identification of expected outcomes.

4.3.4 Paper D

Title: An Empirical Evaluation of System-Level Test Effectiveness for Safety-Critical Software

Authors: Muhammad Nouman Zafar, Wasif Afzal, Eduard Paul Enoiu

Status: Submitted to the 22nd IEEE International Conference on Software Quality, Reliability, and Security 2022 (QRS 2022).

Abstract: Combinatorial Testing (CT) and Model-Based Testing (MBT) are two recognized test generation techniques. The evidence of their fault detection effectiveness and comparison with industrial state-of-the-practice is still scarce, more so at the system level for safety-critical systems, such as those found in trains. We use mutation analysis to perform a comparative evaluation of CT, MBT, and industrial manual testing in terms of their fault detection effectiveness using an industrial case study of the safety-critical train control management system. We examine the fault detection rate per mutant and the relationship between the mutation scores and structural coverage using Modified Condition Decision Coverage (MC/DC). Our results show that CT 3-ways, CT 4-ways, and MBT provide higher mutation scores. MBT did not perform better in detecting 'Logic Replacement Operator-Improved' mutants when compared with the other techniques, while manual testing struggled to find 'Logic Block Replacement Operator' mutants. None of the test suites were able to find 'Time Block Replacement Operator' mutants. CT 2-ways was found to be the least

effective test technique. MBT-generated test suite achieved the highest MC/DC coverage. We also found a generally consistent positive relationship between MC/DC coverage and mutation scores for all test suites.

Chapter 5

Discussion and Limitations

5.1 Discussion

This section provides a discussion on topics of relevance for the thesis. These are the choice of testing techniques and coverage criteria, controlling the test suite size, selection of FBD mutants, the complexity of the SUT, and distribution of faults.

Choice of techniques and coverage criteria: CT, MBT, and industrial manual, scripted testing used in our studies are based on different test design principles. However, one commonality is their applicability at the system level, where system behavior exhibits itself in different combinations that need thorough testing. Additionally, MC/DC is endorsed as a recommended coverage criterion for safety-critical system development by various standards [20] [21] [22]. On the other hand, requirement coverage is one of the de facto coverage criteria in industry. Thus, for coverage criteria measurement, this thesis attempts to research the merits and demerits of the selected test techniques in terms of MC/DC and requirement coverage.

Controlling test suite size: We controlled the size of the test suite for MBT, in the way of removing duplicate test cases (two or more test cases with the same inputs and expected outputs) once test generation terminated. For CT, we did not control the test suite size as the test generation was not terminated until the stopping conditions were met. Due to smaller differences in the number of test cases between different techniques, we do not know for sure if not controlling for the test suite size affects our results, but such an investigation would need new experiments.

Selection and number of FBD mutants: The FBD mutant operators were selected to mimic the actual programmers' mistakes that largely occur due to misinterpretation of requirements. This was done in consultation with a senior developer at Alstom Transport. These actual mistakes were restricted to a limited set of operators, thus limiting the number of FBD mutants. Also, as we write in paper D, one FBD program can be used to generate multiple instances of code related to two or more similar requirements (requirements sharing the same logic), with the result that one mutant operator injected in such FBD programs can produce multiple faults, again limiting the number of generated mutants. Lastly, our endeavor of system-level testing meant that for every mutant, we had to compile, execute and do mutation analysis on a real-world safety-critical system at the software-in-the-loop simulation level. This incurs quite an effort and is drastically different from unit-level mutations in small programs that can be compiled and run against tests in seconds.

System complexity and faults distribution: The train control management system (TCMS) is a high-capacity, infrastructure backbone built on open standard IP-technology that allows easy integration of all control and communication requiring functions on-board the train. TCMS either controls or supervises almost all train functions, such as collecting line voltage, controlling train engines, uploading diagnostic data, opening, and closing doors, etc. All of such functions are built incrementally and tested in different levels of simulation, the most important being the software-in-the-loop and hardware-in-the-loop simulations. A typical TCMS has over 100K lines of code on average, while the lines of code in different functions (like the fire detection subsystem) vary between approximately 4K to 8K lines of code. Note that such code is auto-generated from the FBD programs. We relied on a senior programmer's expertise to create mutants, however, we can also utilize a fault distribution that can enable us to inject more (fictitious) faults in the future.

5.2 Limitations

One of the main limitations of this thesis is the generalizability of the results. We have used one subsystem of the TCMS with the actual number of parameters/signals and constraints defined in the requirement specification, but more case studies are required to generalize the results of this thesis to larger systems.

Another limitation is related to the cost of the testing techniques. The cost model used in this thesis for the evaluation of test generation and execution time of selected techniques is based on the activities followed by industrial practitioners at Alstom Transportation AB, Sweden, and defined in [51]. However, we have not considered the indirect cost and cost of regression testing in our defined cost model, which includes the maintenance cost of the models, test suites and tools, etc., hence, considering such costs can also affect the efficiency results of this thesis. Moreover, the implemented tools based on the proposed framework in this thesis contain particularities for the generation of test scripts that may not be applicable to other domains. However, we have provided some information such as mapping rules and concretization process that can help adoption in other domains.

Chapter 6

Conclusion and Future Work

Focusing on the research goal of this thesis, we have explored and examined the adaption of MBT in industrial settings and proposing a model-based test script generation framework. We have also performed a comparative empirical evaluation of test artifacts (i.e., test cases and test scripts) developed by MBT, CT, and manual testing practices in the industry in terms of cost, test coverage and fault detection effectiveness.

The results of this thesis showed that MBT-generated test cases using a random generator algorithm and edge coverage criterion of the selected tool (i.e., GW) have covered more test scenarios than the test cases generated by other selected coverage criteria and written manually while achieving a similar level of requirement coverage as manually developed test cases. Our studies also showed that the proposed model-based test generation script generation framework can generate test scripts automatically by taking the FSM model and XML file (containing the information about signals interacting between the modules of the system) as inputs.

The empirical evaluation of selected testing techniques has also provided some meaningful implications. CT is found to be the most cost-efficient technique than MBT and manual testing but also requires manual efforts to complete the test suite (i.e., the inclusion of expected output and constraints such as time). Whereas MBT-generated test suites provided the highest MC/DC coverage than CT and manually written test suites and acted as a superset by containing the highest number of such test cases that were also part of the test suites generated by the other testing techniques. It also generated the highest number of unique test cases that showed an observable effect on the MC/DC adequacy. MBT-generated test cases are also found to be more similar and relevant to specification-based manual tests in terms of structure. They can thus complement the manually written test suite to improve its test coverage adequacy.

MBT-generated test suites provided a higher fault detection rate than the manually written test suite, however, it showed a slightly lower fault detection rate than the test suites generated by 3-ways and 4-ways CT. We also found that each testing technique has shown a significant level of sensitivity towards all the mutant operators induced at the FBD level except the operators induced based on timing constraints, where none of the testing techniques were able to detect faults. And lastly, we deduced a positive relationship between the MC/DC coverage and fault detection rate achieved by the generated test suite of a testing technique.

We conclude that there is no clear winner technique with all metrics we have used in our studies. However, in a behavior-driven development environment, the adoption of a hybrid testing strategy with MBT can help in improving test coverage and fault detection effectiveness of manual test artifacts as well as automate the V-model activities.

In the future, we will perform the execution and evaluation of test suites at the hardware-in-the-loop level, explore test suite optimization techniques to minimize the MBT-generated test suite, and perform a comparative evaluation with other techniques such as search-based testing. We also see the implementation and evaluation of online MBT to provide real-time visualization of requirement coverage, code coverage, and values of signals as another possible direction of future work.

Bibliography

[1] Mark Utting and Bruno Legeard. *Practical model-based testing: a tools approach*. Elsevier, 2010.

[2] Gang Yu and Zhong wei Xu. Model-based safety test automation of safety-critical software. In *2010 International Conference on Computational Intelligence and Software Engineering*, pages 1–3. IEEE, 2010.

[3] Dorothy Graham, Rex Black, and Erik Van Veenendaal. *Foundations of software testing*. Cengage, 2019.

[4] Brian Hambling, Peter Morgan, Angelina Samaroo, Geoff Thompson, and Peter Williams. *Software Testing: An ISEB Foundation*. BCS, The Chartered Institute, 2010.

[5] Saswat Anand, Edmund K Burke, Tsong Yueh Chen, John Clark, Myra B Cohen, Wolfgang Grieskamp, Mark Harman, Mary Jean Harrold, Phil McMinn, Antonia Bertolino, et al. An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software*, 86(8):1978–2001, 2013.

[6] Havva Gulay Gurbuz and Bedir Tekinerdogan. Model-based testing for software safety: a systematic mapping study. *Software Quality Journal*, 26(4):1327–1372, 2018.

[7] Li Tan, Jesung Kim, Oleg Sokolsky, and Insup Lee. Model-based testing and monitoring for hybrid embedded systems. In *Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration, 2004. IRI 2004.*, pages 487–492. IEEE, 2004.

[8] Raluca Marinescu, Mehrdad Saadatmand, Alessio Bucaioni, Cristina Seceleanu, and Paul Pettersson. A model-based testing framework for automotive embedded systems. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 38–47. IEEE, 2014.

[9] Jing Guan and Jeff Offutt. A model-based testing technique for component-based real-time embedded systems. In *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 1–10. IEEE, 2015.

[10] Stephan Weißleder and Holger Schlingloff. An evaluation of model-based testing in embedded applications. In *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*, pages 223–232. IEEE, 2014.

[11] Arilo C Dias-Neto and Guilherme H Travassos. A picture from the model-based testing area: concepts, techniques, and challenges. In *Advances in Computers*, volume 80, pages 45–120. Elsevier, 2010.

[12] Eduard Paul Enoiu, Daniel Sundmark, and Paul Pettersson. Model-based test suite generation for function block diagrams using the uppaal model checker. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, pages 158–167. IEEE, 2013.

[13] Johannes Kloos, Tanvir Hussain, and Robert Eschbach. Risk-based testing of safety-critical embedded systems driven by fault tree analysis. In *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, pages 26–33. IEEE, 2011.

[14] Stefano Marrone, Francesco Flammini, Nicola Mazzocca, Roberto Nardone, and Valeria Vittorini. Towards model-driven V&V assessment of railway control systems. *International Journal on Software Tools for Technology Transfer*, 16(6):669–683, 2014.

[15] Deepak A Mathaikutty, Sumit Ahuja, Ajit Dingankar, and Sandeep Shukla. Model-driven test generation for system level validation. In *2007 IEEE International High Level Design Validation and Test Workshop*, pages 83–90. IEEE, 2007.

[16] Mark Utting, Bruno Legeard, Fabrice Bouquet, Elizabeta Fourneret, Fabien Peureux, and Alexandre Vernotte. Chapter two - recent advances in model-based testing. volume 101 of *Advances in Computers*, pages 53 – 120. Elsevier, 2016.

[17] Emil Alégroth, Kristian Karl, Helena Rosshagen, Tomas Helmfridsson, and Nils Olsson. Practitioners' best practices to adopt, use or abandon model-based testing with graphical models for software-intensive systems. *Empirical Software Engineering*, 27(5):1–42, 2022.

[18] Dianxiang Xu, Weifeng Xu, Michael Kent, Lijo Thomas, and Linzhang Wang. An automated test generation technique for software quality assurance. *IEEE Transactions on Reliability*, 64(1):247–268, 2014.

[19] Ana CR Paiva, Daniel Maciel, and Alberto Rodrigues da Silva. From requirements to automated acceptance tests with the RSL language. In *Intl. Conf. on Evaluation of Novel Approaches to SE*. Springer, 2019.

[20] Rob Palin, David Ward, Ibrahim Habli, and Roger Rivett. ISO 26262 safety cases: Compliance and assurance. 2011.

[21] Ron Bell. Introduction to IEC 61508. In *Acm international conference proceeding series*, volume 162, pages 3–12, 2006.

[22] Jean-Louis Boulanger. *CENELEC 50128 and IEC 62279 standards*. John Wiley & Sons, 2015.

[23] Yin Chen, Sven Linder, and Jonas Wigstein. An approach of creating component design specification for safety-related software in railway. In *2019 Annual Reliability and Maintainability Symposium (RAMS)*, pages 1–4. IEEE, 2019.

[24] Arnaud Dupuy and Nancy Leveson. An empirical evaluation of the MC/DC coverage criterion on the HETE-2 satellite software. In *19th DASC. 19th Digital Avionics Systems Conference. Proceedings (Cat. No. 00CH37126)*, volume 1, pages 1B6–1. IEEE, 2000.

[25] Allan L White. Comments on modified condition/decision coverage for software testing [of flight control software]. In *2001 IEEE Aerospace Conference Proceedings (Cat. No. 01TH8542)*, volume 6, pages 2821–2827. IEEE, 2001.

[26] Dong Li, Linghuan Hu, Ruizhi Gao, W Eric Wong, D Richard Kuhn, and Raghu N Kacker. Improving MC/DC and fault detection strength using combinatorial testing. In *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 297–303. IEEE, 2017.

[27] Thomas Bauer, Robert Eschbach, Martin Größl, Tanvir Hussain, Detlef Streitferdt, and Florian Kantz. Combining combinatorial and model-based test approaches for highly configurable safety-critical systems. *Model-based Testing in Practice*, page 9, 2009.

[28] Ge Zhou, Xiang Cai, Chi Hu, Jun Li, Wenxiang Han, and Xiaopin Wang. Research on combinatorial test case generation of safety-critical embedded software. In *International Conference on Mechatronics and Intelligent Robotics*, pages 204–209. Springer, 2017.

[29] Changhai Nie, Baowen Xu, Liang Shi, and Guowei Dong. Automatic test generation for n-way combinatorial testing. In *Quality of Software Architectures and Software Quality*, pages 203–211. Springer, 2005.

[30] Gregory Gay, Matt Staats, Michael W Whalen, and Mats PE Heimdahl. Moving the goalposts: coverage satisfaction is not enough. In *Proceedings of the 7th International Workshop on Search-Based Software Testing*, pages 19–22, 2014.

[31] Laura Inozemtseva and Reid Holmes. Coverage is not strongly correlated with test suite effectiveness. In *Proceedings of the 36th international conference on software engineering*, pages 435–445, 2014.

[32] Xia Cai and Michael R Lyu. The effect of code coverage on fault detection under different testing profiles. In *Proceedings of the 1st International Workshop on Advances in Model-based Testing*, pages 1–7, 2005.

[33] Amanda Schwartz and Michael Hetzel. The impact of fault type on the relationship between code coverage and fault detection. In *Proceedings of the 11th International Workshop on Automation of Software Test*, pages 29–35, 2016.

[34] Bart Broekman and Edwin Notenboom. *Testing embedded software*. Pearson Education, 2003.

[35] Alistair Sutcliffe. *Requirements Analysis for Safety Critical Systems*, pages 149–180. Springer London, London, 2002.

[36] Jan L Rouvroye and Elly G van den Bliek. Comparing safety analysis techniques. *Reliability Engineering & System Safety*, 75(3):289–294, 2002.

[37] M. Maslar. PLC standard programming languages: IEC 1131-3. In *Conference Record of 1996 Annual Pulp and Paper Industry Technical Conference*, pages 26–31, 1996.

[38] Iris Graessler and Julian Hentze. The new V-Model of VDI 2206 and its validation. *at-Automatisierungstechnik*, 68(5):312–324, 2020.

[39] Bohan Liu, He Zhang, and Saichun Zhu. An incremental V-model process for automotive development. In *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*, pages 225–232. IEEE, 2016.

[40] Liu Shuping and Pang Ling. The research of V model in testing embedded software. In *2008 International Conference on Computer Science and Information Technology*, pages 463–466. IEEE, 2008.

[41] Ravi Shanker Yadav. Improvement in the V-Model. *International Journal of Scientific & Engineering Research*, 3(2):1–6, 2012.

[42] Phil McMinn. Search-based software testing: Past, present and future. In *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, pages 153–163. IEEE, 2011.

[43] Richard Hamlet. Random testing. *Encyclopedia of software Engineering*, 2:971–978, 1994.

[44] Alexander Pretschner. Model-based testing. In *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, pages 722–723. IEEE, 2005.

[45] D Richard Kuhn, Raghu N Kacker, and Yu Lei. *Introduction to combinatorial testing*. CRC press, 2013.

[46] Roman Haas, Daniel Elsner, Elmar Juergens, Alexander Pretschner, and Sven Apel. How can manual testing processes be optimized? developer survey, optimization guidelines, and case studies. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1281–1291, 2021.

[47] Gang Yu, Zhong wei Xu, and Jun wei Du. An approach for automated safety testing of safety-critical software system based on safety requirements. In *2009 International forum on information technology and applications*, volume 3, pages 166–169. IEEE, 2009.

[48] Peter H Feiler. Model-based validation of safety-critical embedded systems. In *2010 IEEE Aerospace Conference*, pages 1–10. IEEE, 2010.

[49] Gunwant Dhadyalla, Neelu Kumari, and Timothy Snell. Combinatorial testing for an automotive hybrid electric vehicle control system: a case study. In *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops*, pages 51–57. IEEE, 2014.

[50] Myra B Cohen, Peter B Gibbons, Warwick B Mugridge, and Charles J Colbourn. Constructing test suites for interaction testing. In *25th International Conference on Software Engineering, 2003. Proceedings.*, pages 38–48. IEEE, 2003.

[51] Anne Kramer and Bruno Legeard. *Model-based testing essentials-guide to the ISTQB certified model-based tester: foundation level.* John Wiley & Sons, 2016.

[52] Paul C Jorgensen. *Software testing: a craftsman's approach.* Auerbach Publications, 2013.

[53] Hadi Hemmati, Syed S Arefin, and Howard W Loewen. Evaluating specification-level MC/DC criterion in model-based testing of safety critical systems. In *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, pages 256–265. IEEE, 2018.

[54] Richard Baker and Ibrahim Habli. An empirical evaluation of mutation testing for improving the test quality of safety-critical software. *IEEE Transactions on Software Engineering*, 39(6):787–805, 2012.

[55] Michael Whalen, Gregory Gay, Dongjiang You, Mats PE Heimdahl, and Matt Staats. Observable modified condition/decision coverage. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 102–111. IEEE, 2013.

[56] Allen T Acree, Timothy A Budd, Richard A DeMillo, Richard J Lipton, and Frederick G Sayward. Mutation analysis. Technical report, Georgia Inst of Tech Atlanta School of Information And Computer Science, 1979.

[57] Jonas Boberg. Early fault detection with model-based testing. In *Proceedings of the 7th ACM SIGPLAN workshop on ERLANG*, pages 9–20, 2008.

[58] Aiman Gannous and Anneliese Andrews. Integrating safety certification into model-based testing of safety-critical systems. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, pages 250–260. IEEE, 2019.

[59] Eduard P Enoiu, Adnan Čaušević, Thomas J Ostrand, Elaine J Weyuker, Daniel Sundmark, and Paul Pettersson. Automated test generation using model checking: an industrial evaluation. *International Journal on Software Tools for Technology Transfer*, 18(3):335–353, 2016.

[60] Wan-Hui Tseng and Chin-Feng Fan. Systematic scenario test case generation for nuclear safety systems. *Information and Software Technology*, 55(2):344–356, 2013.

[61] K El-Fakih, Ayman Alzaatreh, and Uraz Cengiz Türker. Assessing test suites of extended finite state machines against model-and code-based faults. *Software Testing, Verification and Reliability*, page e1789, 2021.

[62] Christoph Schulze, Dharmalingam Ganesan, Mikael Lindvall, Rance Cleaveland, and Daniel Goldman. Assessing model-based testing: an empirical study conducted in industry. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 135–144, 2014.

[63] Josip Bozic, Bernhard Garn, Ioannis Kapsalis, Dimitris Simos, Severin Winkler, and Franz Wotawa. Attack pattern-based combinatorial testing with constraints for web security testing. In *2015 IEEE International Conference on Software Quality, Reliability and Security*, pages 207–212. IEEE, 2015.

[64] Vahid Garousi, Alper Buğra Keleş, Yunus Balaman, Zeynep Özdemir Güler, and Andrea Arcuri. Model-based testing in practice: An experience report from the web applications domain. *Journal of Systems and Software*, 180:111032, 2021.

[65] Marvin Rausand. *Reliability of safety-critical systems: theory and applications*. John Wiley & Sons, 2014.

[66] Arend Aerts, Michel Reniers, and Mohammad Reza Mousavi. Model-based testing of cyber-physical systems. In *Cyber-Physical Systems*, pages 287–304. Elsevier, 2017.

[67] Colin Robson and Kieran McCartan. *Real world research: a resource for users of social research methods in applied settings*. Wiley, 2016.

[68] Conradi and Hofmann. *Empirical Methods and Studies in Software Engineering*. Springer Berlin Heidelberg, 2003.

[69] Muhammad Nouman Zafar, Wasif Afzal, Eduard Paul Enoiu, Athanasios Stratis, Aitor Arrieta, and Goiuria Sagardui. Model-based testing in practice: An industrial case study using graphwalker. In *Innovations in Software Engineering Conference 2021*, February 2021.

[70] Muhammad Nouman Zafar, Wasif Afzal, Eduard Paul Enoiu, Athanasios Stratis, and Ola Sellin. A model-based test script generation framework for embedded software. In *The 17th Workshop on Advances in Model Based Testing*, February 2021.

[71] Muhammad Nouman Zafar, Wasif Afzal, and Eduard Paul Enoiu. Evaluating system-level test generation for industrial software: A comparison between manual, combinatorial and model-based testing. In *The 3rd ACM/IEEE International Conference on Automation of Software Test 2022*, March 2022.