

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

Muhammad Nouman Zafar^a, Wasif Afzal^b and Eduard Paul Enoiu^c

Mälardalen University, Sweden

{muhammad.nouman.zafar, wasif.afzal, eduard.paul.enoiu}@mdu.se

Keywords: System-Level Tests, Safety-Critical Software, Fault Detection Effectiveness.


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 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.


1 INTRODUCTION


Software controls safety-critical functions of systems in different domains, e.g., in avionics and vehicular. However, the failure of software in such systems directly affects the physical world. Therefore, failures in safety-critical software can lead to substantial risk to the safety of human lives, serious environmental damage, and severe economic problems. Engineering safety-critical systems typically require a certain degree of certification according to safety standards and defined processes for thoroughly analyzing the system requirements, together with software testing to ensure its reliability.

The analysis of the preliminary system requirements is performed by one of the qualitative or quantitative safety analysis techniques such as expert analysis, failure mode and effect analysis, reliability block diagrams, and fault tree analysis (Rouvroye and van den Blik, 2002). After analyzing system requirements, safety-critical functions are usually im-

plemented in specific industrial control software systems. For example, Programmable Logic Controllers (PLCs) support multiple programming languages for the development of industrial applications and have been widely adopted in several domains. The testing of such software is then carried out to ensure adequate functional and non-functional operations of the system according to specific system requirements. However, testing such systems is costly. As a solution to reducing the cost of testing and assuring the reliability of such systems, several techniques for automated test generation such as model-based testing (MBT), and combinatorial testing (CT) exist. Besides, manual testing is still considered a prevailing technique for the testing of real-world industrial applications (Taipale et al., 2011), with some evidence suggesting that both technical and non-technical skills are required for effective fault detection (Sánchez-Gordón et al., 2020). There is a scarcity of research and consequently empirical evidence into the fault-detection effectiveness of automated test generation techniques and industrial manual testing, especially if the aim is to generate test cases at the system level of a safety-critical system. Thus, the use of automated test generation alongside industrial manual testing re-

^a  <https://orcid.org/0000-0001-8746-7209>

^b  <https://orcid.org/0000-0003-0611-2655>

^c  <https://orcid.org/0000-0003-2416-4205>

quires a thorough evaluation of the fault detection capabilities in an industrial setting.

This motivated us to analyze the fault detection effectiveness of manual and two well-known automated test generation techniques (i.e., CT and MBT) through mutation analysis, at the system level of the safety-critical train control management system (TCMS) developed by Alstom Transportation AB. We have also analyzed the type of detected faults induced by different mutation operators to identify the sensitivity of each technique toward each type of injected fault. In addition, we performed a comparative analysis between the mutation scores and coverage levels¹ of each test suite. Based on our goal, we have formulated the following research questions:

- **RQ1.** How effective are CT, MBT, and manual testing techniques in detecting injected faults?
- **RQ2.** How sensitive are CT, MBT, and manual system-level testing to different types of injected faults?
- **RQ3.** What is the relationship between MC/DC coverage and mutation scores achieved by CT, MBT, and manual testing at the system level?

The rest of the paper is organized as follows: Section 2 provides an overview of the background and related work, Section 3 presents the methodology of this study including an overview of the System Under Test (SUT), a description of the development process and mutation injection, test suite creation, deployment of the SUT and execution of test scripts, and evaluation metrics. Section 4 describes the results, Section 5 provides a brief discussion on results, Section 6 deals with the validity threats, whereas the conclusion and future are provided in Section 7.

2 BACKGROUND AND RELATED WORK

This section provides necessary background information on relevant topics and a summary of related work.

2.1 Safety-Critical Software and PLCs

PLCs are used to control the safety-critical functions of an embedded system application. PLCs are specially designed industrial computers connected via a network of input and output modules using a data bus. These PLCs communicate with each other,

¹We use MC/DC since several standards, e.g., EN 50129, suggest its use in safety-critical software system development

sensors, and other subsystems to control the safety-critical functions of an embedded system. Functional Block Diagram (FBD) is one of the five standard languages defined by the International Electrotechnical Commission (IEC) (Maslar, 1996) to implement safety-critical applications for PLCs and has been widely adopted by industrial practitioners in domains such as railways, nuclear power plants, and avionics (Schwartz et al., 2010).

FBD is a graphical notation language that contains ten function block groups stated in IEC 1131-3 (Maslar, 1996) (e.g., arithmetic, logic, comparison and timer operations). These function block groups are responsible for different types of operations between the states of input and output variables. Function elements/blocks from these groups are connected through input/output variables to represent the behavior of an FBD program. The developed FBD programs are then compiled into the source and machine code using platform-dependent tools provided by PLC vendors.

2.2 System-Level Test Generation Techniques

Automatic test case generation techniques exploit software artifacts as inputs to generate test cases to achieve better effectiveness and efficiency (Anand et al., 2013). Multiple test generation techniques exist such as random (Hamlet, 1994), search-based (McMinn, 2011), model-based (Utting and Legeard, 2010), amongst others to generate high-quality test suites to validate a system, mostly at the functional and structural level. But different studies (e.g. (Li et al., 2017)) report advantages of CT and MBT for testing safety-critical software at the system level. There exist other studies (e.g., (Zafar et al., 2022)) that have provided a comparative evaluation of CT and MBT with manual testing in practice in terms of coverage criteria.

CT generates a covering array in the form of interaction possibilities of values selected from the input parameters of a system (Nie and Leung, 2011). Each row of a covering array represents a specific test case to validate the functional requirements of a system using the generated combination of input parameters. CT provides a t-ways test strategy to generate relevant and finite number of input interactions. The t-ways testing strategy generates different test cases in a test suite depending on the value of t , the interaction strength. It ensures the generation of unique test sequences by covering each combination based on t-ways (2, 3, and 4-ways being the most common) input parameter interactions at least once.

MBT (Tian et al., 2021) generates test cases from an explicit model representing the SUT. For instance, a model created using a finite state machine (FSM) consists of nodes and edges. Nodes represent the states of the SUT, whereas edges represent the transitions between the states containing the guard conditions depicting the functional and non-functional behavior of the SUT in terms of Boolean constraints. After the creation of the model, a variety of abstract test cases can be generated from the same model of a SUT depending on selected coverage criteria (e.g., edge, node, and requirement) and generator algorithms (e.g., random, weight random, A-star). These abstract test cases are then converted into concrete, executable test scripts to validate a system.

2.3 Modified Condition Decision Coverage

Testing coverage criteria are used to determine the extent to which the design and implementation structure has been exercised. They evaluate the quality of a test suite by measuring, e.g., the area of a code accessed, number of states visited, number of logical predicates covered, or path traversed by a test suite (Lindström et al., 2018). Some known structural coverage criteria are condition coverage, decision coverage, and modified condition decision coverage (Hemmati, 2015). However, MC/DC coverage is recommended by certain standards (i.e., ISO 26262, EN 50128, and EN 50657) to evaluate the structural coverage for safety-critical systems. Multiple studies (e.g., (Li et al., 2017), (Vilkomir et al., 2017b)) have shown the effectiveness and usability of the MC/DC to evaluate (and generate) test cases. It ensures the coverage of each parameter in a program i.e., each condition, decision, entry, and exit point of a program as well as the independent effect of a condition on a decision at least once. Hence, to evaluate a test suite in terms of MC/DC each of the above-mentioned parameters should be examined (Johnson et al., 1998). The testing of safety-critical functions of TCMS needs to follow EN 50128 and EN 50657 safety standards; they suggest MC/DC as the code coverage metric, leading us to examine the quality of test suites in this paper in terms of MC/DC coverage.

2.4 Mutation Analysis

While there are multiple studies showing that test suite adequacy can be assessed using different coverage criteria (Hemmati, 2015), there is evidence showing that these criteria are not enough to evaluate the quality of a test suite in terms of fault detection ef-

fectiveness. Multiple factors can affect the ability of a test suite to detect faults (Schwartz and Hetzel, 2016). Mutation analysis is one of the techniques that can be used to examine the ability of a test suite in terms of fault detection effectiveness, especially in cases where naturally-occurring faults are not available.

Mutation analysis is an error-based approach used to evaluate the quality of test suites by measuring the number of detected faults that were induced in the real system (Acree et al., 1979). It involves the creation of different versions of an original program by injecting a small fault in each version. These faulty versions of an original program are also known as mutants. Each mutant represents a common and typical syntactical error in a programming language or a logical error produced due to the misinterpretation of a requirement by a developer. These mutants can be categorized as equivalent and non-equivalent or stubborn mutants (Yao et al., 2014). Equivalent mutants can not be killed by any of the test cases in a test suite because of their identical behavior to the original program. Whereas a non-equivalent mutant can be killed with one or more potential test cases as they exhibit different behavior than the original program. After the creation of mutated versions, test suites are designed with specific testing techniques and executed on the programs (i.e., original and mutated programs) to check if the mutant is killed or not. A mutant is said to be killed by a test suite t if the test results of a test suite executed on mutated and original programs show a contradiction. On the other hand, if the results of mutated and original programs are identical, then the mutant is considered alive (M_A). The mutation score is then calculated based on the total number of mutants (M_T) and the number of mutants killed (M_K). The mutation score can be calculated using either an output-only oracle (i.e., strong mutation) or a state change/internal oracle (i.e., weak mutation) against the set of mutants. However, in this study, we have used strong mutation for the evaluation of the test suites.

2.5 Related Work

Mutation testing is used to assess the test suites as well as to validate software in different domains (Petrovic et al., 2018). Multiple state-of-the-art studies (Oh et al., 2005), (Hierons and Merayo, 2009), (Lindström et al., 2018), (Enoiu et al., 2016b), (Delgado-Pérez et al., 2018), (Baker and Habli, 2012), (Ramler et al., 2017), (Enoiu et al., 2016a) have explored and examined mutation testing using platform-dependent languages (e.g., C, C++,

JAVA, Python, and FBDs) for generating test suites and evaluating testing techniques and coverage criteria. Since the focus of our paper is on mutation analysis for test coverage and testing technique evaluation, rather than using it for test generation, below we summarize the papers falling in our focus areas only.

Pedro et. al (Delgado-Pérez et al., 2018) used mutation testing to evaluate the structural coverage criterion and to determine the weakness in the testing practices using an industrial case study. The results showed the potential failures caused by the test suite generated by the branch coverage criterion and improvements in the test suites derived from the mutation analysis of the faults and test results. Similarly, an empirical evaluation has been conducted to improve the test quality of a test suite using mutation testing in (Baker and Habli, 2012). They also showed the comparison of mutation analysis with traditional structural coverage analysis and manual peer review in terms of identifying deficits in test suites and testing activities. The results demonstrate that mutation testing is the most effective in detecting issues in test suites and testing activities than manual review and structural coverage analysis. However, the results also showed that mutation testing can not replace manual review but can be used as a complement to it.

Rudolf et. al (Ramler et al., 2017) investigated the fault detection effectiveness of unit-level tests by applying the mutation analysis in the context of safety-critical systems. They also examined the applicability of mutation analysis in terms of improving tests generated at the unit level by the MC/DC coverage criteria. The evaluation showed the deficiencies which were hard to find in the test suites and provided empirical evidence for improving test suites using mutation testing. The fault detection efficiency and effectiveness have also been determined and compared for the specification and implementation-based testing using an industrial case study in (Enoiu et al., 2016a). The mutation analysis of the results indicates that implementation-based testing is an effective testing technique and provided a higher structural coverage than test suites created by specification-based testing.

The aforementioned studies used mutation analysis to generate test suites as well as to compare and evaluate different testing techniques and coverage criteria in terms of fault detection effectiveness. However, we have found only a limited set of studies ((Shin et al., 2012), (Shin et al., 2016), (Charbachi et al., 2017), (Ahmed et al., 2020)), which have evaluated the automated testing techniques using FBD-level mutation analysis in an industrial setting. Thus, we examine the quality of generated test suites gen-

erated by MBT and CT in the industry and provide a comparative analysis of these techniques with manual practices, with the aim to improve dynamic testing of safety-critical systems.

3 METHODOLOGY

The methodology that we have developed and followed throughout our study has been presented in Figure 1. It consists of five main steps; (1) analysis of system requirements and test specification (i.e., Steps (1.1), (1.2), and (1.3)), (2) development of mutated versions of the original program (i.e., (2.1), (2.2), and (2.3)), (3) creation of test suites using each testing technique (i.e., (3.1) and (3.2)), (4) deployment and execution of programs and test scripts to generate test results (i.e., (4.1), (4.2), and (4.3)), (5) evaluation of test scripts based on test results.

3.1 Description of SUT

We considered the fire detection system as a case study from an ongoing project at Alstom Transportation AB. The fire detection system is a subsystem of the Train Control Management System (TCMS) for MOVIA² which is a family of vehicle products developed at Alstom Transportation and operational in various metro trains across the globe. TCMS is an integral part of the complex distributed control system of a train used to control and manage all the operational functions of a train by communicating with different devices and subsystems via various networks i.e. Ethernet Consist Network (ECN), Multi-function Vehicle Bus (MVB), etc. (Zafar et al., 2021a). All the safety-critical functions of a train are controlled by Modular Input/Output-Safe (MIO-S) and Central Control Unit-Safe (CCU-S) devices connected via the MVB network. The fire detection system in TCMS relates to a safety-critical function used to detect two types of fire in the cabs of a train, i.e., internal, and external. It uses two instances of Fire Detection Control Units (FDCUs) connected with the smoke and fire sensors. Each FDCU can have two states (i.e. *Master* and *Slave*) and only be considered as a '*Master*' if it holds the value of its signals as true, whereas the sensors are responsible to transmit signals indicating fire to the FDCUs. The MIO-S device of TCMS receives the signals from the FDCUs and reports these signals to CCU-S. CCU-S computes the logic based on the functional requirements and reports the type of fire to

²<https://www.railway-technology.com/projects/bombardier-movia-metro-cars/>

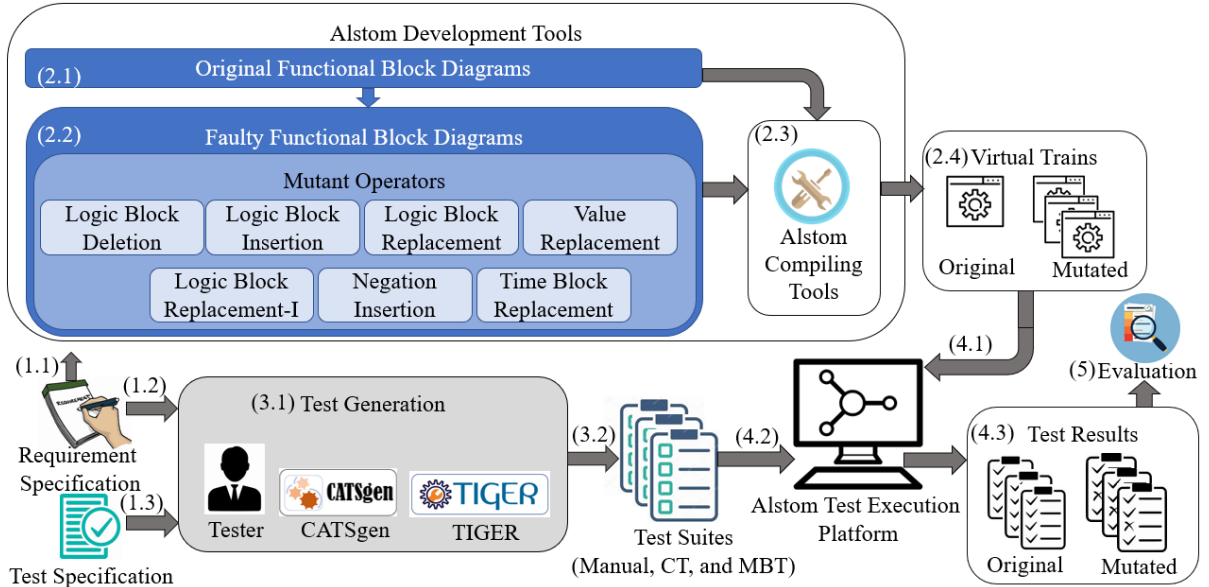


Figure 1: An overview of the experimental methodology.

MIO-S as a corresponding output signal. The MIOS receives the signal to light a LED indicating the type of fire on the driver’s desk.

3.2 Requirements and Test Specification Analysis

We have thoroughly analyzed the requirements of the industrial case study to understand the functional and non-functional behavior of the system. It includes the identification of input & output signals used to communicate with the sensors, PLCs, and TCMS, data types of the identified signals, constraints such as the system’s response time, and arithmetic/Boolean operators that should be used to develop the desired behavior of the system using FBDs. On the other hand, the test specification analysis helped us create the complete SUT model for the generation of the test suite using MBT. It also helped us understand the test objectives, different test scenarios, and behavior of the system from a tester’s perspective. Software engineers and testers at Alstom Transportation use the logical signal names to specify the requirements and test cases whereas the actual system uses one or more technical signal names for its regular and safety-critical operations. Hence, the analysis of requirements and test specifications also provided us with traceability between logical and technical signal names in addition to an understanding of the development process.

3.3 Development of SUT and Mutation Injection

The general principle of mutation analysis is to examine the detection of injected faults in an original program. These faults are injected based on some mutant operators that can be used to mimic a programmer’s common mistakes. So, we have used the original FBD program of the industrial case study developed by a developer at Alstom Transportation and created a set of mutants based on selected mutant operators manually. The selection of mutant operators is carried out by thoroughly reviewing the previous studies from the literature (Oh et al., 2005) (Enoiu et al., 2016b) (Shin et al., 2012) that applies mutation analysis specifically for the evaluation of testing technique, test coverage, or to generate test suites using FBD programs. Moreover, the development of safety-critical FBD programs at Alstom Transportation requires only specific FBD operators. So, by considering the safety-critical industrial case study and FBD-specific faults, we have used seven mutant operators as follows:

- *Logic Block Deletion Operator (LDO)*: to delete a logical block from the FBD program (e.g., deleting AND block).
- *Logic Block Insertion Operator (LIO)*: to insert another logical block between the logical blocks or the input signals of the FBD program (e.g., inserting AND block between the output of two OR blocks).

- *Logic Block Replacement Operator (LRO)*: to replace a logical block of the FBD program with another logical block of the same category (e.g., replacing the AND block with the OR block).
- *Logic Block Replacement Operator-Improved (LRO-I)*: to replace the logical block of the FBD program with a logical block of the same category and another logical block with Boolean input (e.g., replacing OR with RS).
- *Negation Insertion Operator (NIO)*: to insert the negation block at the inputs or outputs of other logical blocks.
- *Time Block Replacement Operator (TRO)*: to replace the timer block with another timer block (e.g., replacing TOF with TON).
- *Value Replacement Operator (VRO)*: to replace the constant value of a variable provided to a block with another value (e.g., replacing the timer variable from 3s to 6s).

After injecting the faults based on the selected mutants, we used the Alstom Transportation-specific compiling tools to generate the builds of a train containing the mutated program. We used these builds to generate the simulations of a train, also known as virtual trains, to execute the test scripts at the software-in-the-loop level.

3.4 Test Suite Creation

To evaluate the test suite developed using manual testing, we used the manually created test suite by a tester at Alstom Transportation for the selected subsystem. Whereas, for MBT and CT, we have utilized automated test script generation tools i.e. TIGER (Model-Based Test sCrIpt GenErAtion fRamework) (Zafar et al., 2021b) and CATSgen (Covering Array Test Script generator) to generate the test suites, respectively. Both automated tools are based on different abstract test case generation tools but use a similar procedure, and format of the XML file containing the information about the signals (i.e., data type, logical and technical signal names) to generate the executable test scripts. Moreover, the test scripts developed by the selected techniques are implemented in the C# language. A brief description of the activities and tools for test suite generation of each testing technique is given in the subsequent subsections.

3.4.1 Manual Test Suite Creation

The testers at Alstom Transportation follow EN 50128 and EN 50657 safety standards and regulations

to create the test suites based on Equivalence Partitioning (EP) and Boundary Value Analysis (BVA) testing techniques. However, in some cases, MC/DC coverage criterion is also used for the creation the test suites for testing complex systems. The test cases are written in natural language and consist of a set of test steps specifying test inputs for the system, expected output, and response time according to each requirement specified in the requirement specification. Requirement coverage is considered a de facto criterion at Alstom Transportation for test suite development to ensure that each requirement has been covered and executed by the test cases. After the creation of test cases, Alstom Transportation-specific libraries are used to write the test scripts manually.

3.4.2 Model-based Test Suite Generation

For model-based test suite generation, we have utilized GraphWalker³ studio version to create the FSM model of the SUT and provided the model to TIGER along with an XML file to generate C# implemented test scripts. TIGER uses the CLI version of GraphWalker to generate the abstract test cases in JSON format by traversing through the model elements based on the selected generator algorithm (e.g., random, quick random, etc.) and coverage criteria (edge, vertex, requirement, etc.). It contains the implementation of some defined mapping rules for logical and technical signal names as well as information specific to Alstom Transportation's testing framework (i.e., configurations, classes, and methods). After the generation of abstract test cases, it processes the data in a JSON file and utilizes the mapping rules along with testing framework-specific information to generate concrete test scripts. Hence, we generated the test suite by selecting the 'random' generator algorithm and 100% edge coverage criteria using TIGER.

3.4.3 Combinatorial Test Suite Generation

There exist multiple combinatorial test generation tools to generate test cases using different algorithms and combinatorial interaction strengths (Khalsa and Labiche, 2014). However, these tools can only be used to generate abstract test cases in the form of a covering array. To execute the test cases on the Alstom Transportation-specific testing framework, these test cases need to be concretized and implemented in the C# language. Hence, to generate the combinatorial-based test suites, we have developed our own CT test script generator called CATSgen based on a state-of-the-art combinatorial test gen-

³<https://github.com/GraphWalker/graphwalker-project/wiki>

eration tool known as CAgen⁴. CAgen is an open-source tool and available in two versions (i.e. online web GUI and offline command-line) with comparatively high performance than other combinatorial test generation tools (Wagner et al., 2020). It provides three state-of-the-art meta-heuristic search algorithms (i.e. FIPOG, FIPOG-F, FIPOG-F2) to generate the test cases based on the t-ways testing strategy. We have considered the logical names of all the signals specified in the requirement specification as input parameters and utilized the web GUI online version of CAgen and selected the FIPOG heuristic algorithm to generate the test cases. We have also provided the test redundancy value ‘1’ with the ‘randomization of don’t care values’ for the generation of non-redundant test cases and limited the interaction strength to 2, 3, and 4-ways to avoid combinatorial explosion (Ramler et al., 2012).

After the creation of the test cases, we exported the generated test cases in an excel file and added the expected outputs and timing constraints against each test case manually by thoroughly analyzing the requirements of the system. Then we provided the exported file along with an XML file as input to CATSgen. CATSgen contains the implementation of mapping rules, similar to TIGER, to map the logical signal names and their respective values to the technical signal names. Moreover, it also contains the implementation details specific to the Alstom Transportation’s testing framework and libraries (e.g., configurations, classes, methods, etc.) to generate the executable test scripts. Hence, CATSgen extracted the data from the excel and XML files, used the mapping rules along with implementation details, and generated the test scripts in the C# language.

3.5 Deployment and Execution of the SUT and Test Scripts

After the generation of virtual trains and test scripts, we deployed the virtual trains on a laptop containing the software compatible with the Alstom Transportation-specific testing framework and test simulation platform for TCMS. Moreover, we used the Alstom Transportation-specific libraries and configuration files to set up the testing environment in a project using Visual Studio 2019 and executed the test scripts to generate test results in the form of test verdicts. The generated test verdicts contained passed and failed test steps that can be used to identify the detection of a fault produced by a mutant operator in a program.

⁴<https://matris.sba-research.org/tools/cagen/#/about>

3.6 Evaluation

We have used two metrics for the evaluation of each test suite based on our formulated research questions as follows:

3.6.1 Mutation Score

A mutation score can be calculated for a test suite by using the total number of mutants $P(M_T)$ created for a program P and the total number of mutants killed $T(M_K)$ by the test suite T in the defined formula:

$$MutationScore (\%) = T(M_K) / P(M_T) \times 100$$

The mutation score can be used to deduce the fault detection effectiveness of each test suite. The percentage of mutants score shows the number of mutants in percentage detected by a test suite, for example, a test suite with a maximum mutation score (i.e. 100) depicts that 100% of the mutants are detected by that particular test suite. Hence, we used the mutation score to compare and evaluate the fault detection effectiveness of test suites developed by each testing technique.

3.6.2 MC/DC Coverage

We used the MC/DC coverage criteria to examine the structural coverage and analyze the relationship between the MC/DC and mutation score in terms of fault detection effectiveness of a test suite. There is no standard formula to calculate the MC/DC adequacy and different existing tools use different principles for measuring the MC/DC of test suites (Vilkomir et al., 2017a). However, we calculated each coverage parameter presented for MC/DC in Section 2.3 and used the following formula, similar to the one used in (Vilkomir et al., 2017b), to measure the MC/DC of each test suite:

- Conditions having all possible outcomes (C) % = (No. of conditions having all possible outcomes / Total no. of conditions) x 100
- Decisions having all possible outcomes (D) % = (No. of decisions having all possible outcomes / Total no. of decisions) x 100
- Conditions independently affecting a decision (AD) % = (No. of conditions independently affected decisions / Total no. of conditions affecting decisions) x 100
- Entry and exit points of program invoked (E) % = (Sum of no. of invoked entry and exit points / Sum of total no. of entry and exit points) x 100

Hence, the overall MC/DC coverage in percentage for each test suite is calculated as an average of the individual MC/DC conditions of C, D, AD, and E.

4 RESULTS

In this section, we provide the experimental results of our study in terms of fault detection effectiveness for each test suite using mutation score, the sensitivity of each test suite towards mutation operators, and an analysis of the relationship between MC/DC and mutation score in detecting faults.

4.1 RQ1: Fault Detection Effectiveness of Test Suites

To measure the fault detection effectiveness of each test suite, we created 50 mutants of the original FBD program based on mutation operators as described in Section 3.3 and calculated the mutation score for each test suite. The mutation scores are shown in Column 5 of Table 1. It is important to mention here that we have considered only non-equivalent mutants in our results and excluded the equivalent mutants⁵ after carefully examining the test results and alive mutants manually. For example, we added an XOR between two operators in one of the LIO mutants, and none of the test suites detected this fault. On examining this mutant, we found out that it had an apparent effect at the internal level, but that effect did not propagate towards the overall output of the program. Hence, we declared this mutant an equivalent and excluded it from the results. Table 1 shows the result of each test suite in terms of mutation score, and the total number of killed and alive mutants. Table 2 depicts the number of common mutants killed and alive between the pairs of test suites. Whereas, Figure 2⁶ illustrates the overlaps and differences between alive mutants of each test suite.

The results show that CT-generated test suites provide the highest mutation scores by detecting 90% of the mutants when using 3-ways and 4-ways interaction strength while requiring 1140 and 1680 seconds as an average execution time per mutant, respectively. In contrast, CT 2-ways, manual, and MBT achieved 82%, 86%, and 88% mutation scores respectively, and 780, 600, and 2760 seconds of average execution time per mutant, respectively. Moreover, the number of undetected mutants in manual, CT 2-ways, CT 3-ways, CT 4-ways, and MBT were 7, 9, 5, 5, and 6, respectively. We observed that MBT generated the highest number of test cases in a test suite and required the highest execution time, on average per mutant, but still provided a slightly low mutation score than 3-ways and 4-ways testing strategy. In addition, we also

⁵The mutants that do not change the program behavior.

⁶The Venn diagram is created in an online tool 'Meta-Chart' <https://www.meta-chart.com/venn#/data>

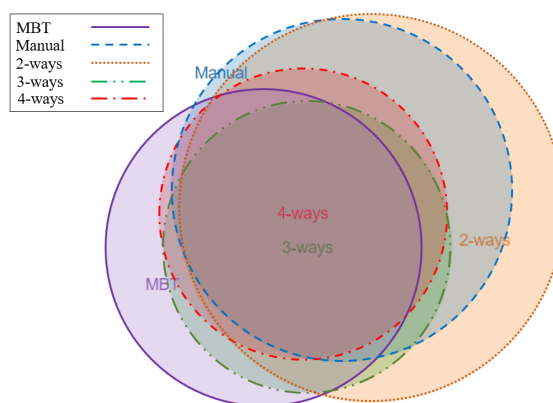


Figure 2: Venn diagram representing the overlaps and differences between alive mutants.

examined the test verdicts of the MBT-generated test suite and found that, due to duplicate test cases in the MBT-generated test suite, if a test case in the MBT-generated test suite detects a fault, its identical test case(s) also show the presence of the fault in the mutated program. In our industrial context, each failed test step in a test case requires a maximum waiting time for a signal response specified in the requirements. Similarly, in the case of other test suites, the number of test cases detecting a mutant and the waiting time for a signal response have a significant effect on the execution time. However, the number of MBT-generated test cases in a test suite can be minimized by removing these identical test cases, which consequently will reduce the execution time too.

The analysis of the data shown in Table 2 and Figure 2 suggests an similar number of killed mutants whereas only 4 common alive mutants were found among all the test suites. We have also examined these alive mutants and observed that 2 out of these 4 mutants were affecting a part of the code that could not be invoked by the generated test cases' inputs and requires inputs from another subsystem upon integration. However, test suites developed using Manual, 2-ways, 3-ways, 4-ways, and MBT left 3, 5, 1, 1, and 2 mutants alive, respectively⁷. Hence, all the test suites achieved a reasonably high level of mutation scores, within the range of 82% to 90%. However, the test suites generated by CT 3-ways and 4-ways provided higher fault detection rates than the test suites generated by other testing techniques. Moreover, each technique missed the generation of some signal combinations that could be used to achieve a mutation score of 100%. However, we still found manual testing as a better-performing technique in terms of average exe-

⁷Some of these are unique across all the test suites, while some have overlaps among only a subset of the test suites.

Table 1: Mutation score of each testing technique.

Techniques	No. of Test Cases	Mutants Killed (M_K)%	Mutants Alive (M_A)%	Mutation Score%
Manual	17	43	7	86
CT 2-ways	10	41	9	82
CT 3-ways	22	45	5	90
CT 4-ways	50	45	5	90
MBT	150 (39 after excluding identical test cases)	44	6	88

Table 2: No. of common killed and alive mutants between the pairs of test suites.

Techniques	CT 2-ways	CT 3-ways	CT 4-ways	MBT
	(M_K / M_A)	(M_K / M_A)	(M_K / M_A)	(M_K / M_A)
Manual	41/7	42/4	43/5	41/4
CT 2-ways	N/A	41/5	41/5	39/4
CT 3-ways	-	N/A	41/5	43/4
CT 4-ways	-	-	N/A	43/4

cutation time per mutant while achieving a 86% of mutation score. A summary of our observations regarding alive mutants across all test suites is as follows:

- The total number of mutants not detected by manual testing, 2-ways, 3-ways, 4-ways, and MBT were 7, 9, 5, 5, and 6, respectively. 4 of these alive mutants were common among all the test suites.
- The manual test suite did not detect 3 alive mutants that were also included in the subset of alive mutants in CT 2-ways, whereas 1 mutant amongst the 3 was also not detected by the 2-ways and 4-ways generated test suites.
- The use of CT 2-ways generated test suite did not detect 1 unique alive mutant which was killed by each of the other test suites.
- CT 3-ways and 2-ways generated test suites had 1 alive mutant which was not killed by either of the test suites.
- Lastly, 2 unique mutants were not detected by the MBT-generated test suite.

4.2 RQ2: Sensitivity of Test Suites to Specific Mutation Operators

To examine the type of faults prone to be detected by each test suite, we calculated the mutation scores as per each mutation operator. We also developed a bar graph based on the results to analyze the breakdown of mutation scores of each test suite as shown in Figure 3.

Figure 3 shows some meaningful implications based on the mutants killed by different test suites. All test suites detected each of the mutants injected

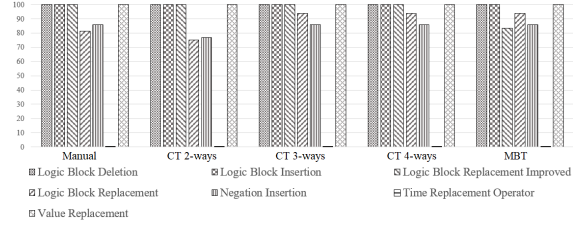


Figure 3: Percentage of mutants killed by each test suite per operator.

by 3 out of 7 mutation operators i.e., LDO, LIO, and VRO, and provided 100% mutation scores for these mutation operators. On the other hand, none of the test suites detected any mutant induced in the original program based on the TRO mutation operator, consequently providing 0% mutation scores. In the case of LRO-I, all the test suites achieved 100% mutation scores except MBT, which achieved only 83%. For the NIO-based mutants, all the test suites achieved 86% mutation scores except 2-ways, which achieved 77%. Similarly, 3-ways, 4-ways, and MBT attained a similar mutation score of 93% by detecting mutants injected based on LRO, whereas 2-ways and manual achieved 75% and 81%, respectively.

Based on these results, we observe that for each mutation operator except TRO, each testing technique did not generate such combinations (i.e. combinations of inputs invoking the faulty area of the code affected by the alive mutant) that could be used to reach 100% mutation score. In order to detect TRO-based mutants, each test suite requires special test cases targeting the basic integrity (i.e., ensuring the starting states of the system) while entering a state within specified *time*, which can be used to validate the timing related requirements of the system.

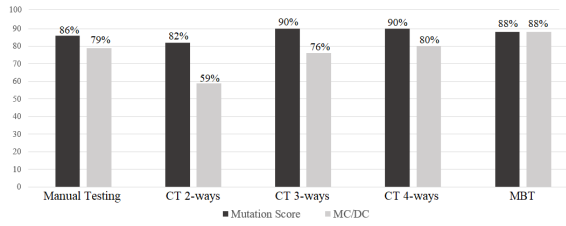


Figure 4: Mutation score and MC/DC of each test suite in percentage.

Hence, the results show that the CT 3-ways and 4-ways generated test suites are the most effective in detecting all types of faults except the faults related to TRO. MBT’s mutation score closely follows, where it achieved an equal mutation score in detecting all types of faults when compared to CT 3-ways and 4-ways, except faults related to LRO-I. CT 2-ways-generated test suites are least effective in detecting NIO and LRO-based mutants due to the generation of less number of input combinations having lower interaction strength.

4.3 RQ3: Relationship Between MC/DC Coverage and Mutation Score

Figure 4 presents the mutation score and MC/DC coverage of each test suite. It shows that the MBT-generated test suite provided the highest MC/DC coverage, i.e., 88%. The test suites developed using CT 2-ways, 3-ways, 4-ways, and manual testing techniques provided 59%, 76%, 80%, and 79% of MC/DC coverage respectively. The test suite generated by MBT provided an equal mutation score as its MC/DC coverage, whereas manual, 2-ways, 3-ways, and 4-ways generated test suites provided higher mutation scores as compared to the MC/DC coverage achieved by the respective test suites. We also observed that the differences between the mutation scores and MC/DC coverage in 2-ways and 3-ways generated test suites were greater than for the test suites generated by 4-ways and manual testing. Moreover, regardless of its high MC/DC, MBT shows a slightly lower mutation rate when compared with CT 3-ways and 4-ways techniques.

To thoroughly analyze the relationship between MC/DC coverage and mutation scores, we examined the breakdown of MC/DC coverage achieved by each test suite according to the selected parameters as shown in Figure 5, the SUT, as well as the killed and alive mutants. The analysis showed that if a system shares two or more similar requirements, then one FBD program can be used to generate two instances of the code. Therefore, each mutant induced at the FBD level can affect different areas of the code (i.e.

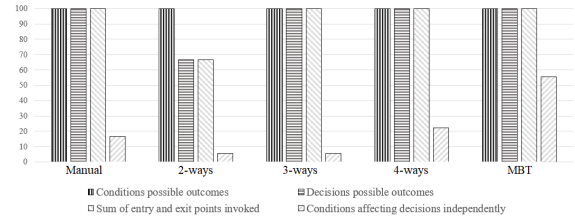


Figure 5: Breakdown of MC/DC according to selected parameters.

induce more than one fault) and it may be possible that test suites with lower MC/DC can not detect each fault induced per mutant. Similarly in our case, a fault induced at the FBD level affected the behavior of the selected program at the system level where the combinations of input signals have a significant effect on the output of the system. Hence, the test suites developed using CT with higher interaction strength i.e., 3-ways and 4-ways and manual testing techniques contained such combinations, which were required to achieve an adequate mutation score at the system level. Whereas, MBT has provided higher MC/DC coverage by generating such test cases, especially targeting the MC/DC parameter ‘coverage of all the conditions having an independent effect on a decision at least once’, where other techniques gave relatively lower coverage percentages. Based on the graphs presented in Figure 4 and Figure 5, we also observed a generally positive relationship between MC/DC coverage and mutation score and argue that test suites achieving an adequate level of MC/DC coverage also provide higher mutation scores.

5 DISCUSSION

We see from our results that a high fault detection rate can be achieved using the higher interaction strength of CT. Results are similar to some previous studies, e.g., (Bures and Ahmed, 2017), (Petke et al., 2013). However, CT has an increasing cost of completing the test suite with expected outputs and timing constraints according to the system requirements. On the other hand, MBT generated a complete set of test suites containing inputs, expected outputs, and timing constraints from the model conforming to system requirements while achieving an 88% fault detection rate. Moreover, we also found the MBT-generated test suite is complete (i.e., contained expected output, and timing constraints) and ready to be executed as well as similar to specification-based manual testing used in the industry.

The analysis of different mutant operators illustrates that all the test suites do not detect faults re-

lated to time constraints and thus special test cases should be included to target testing of timing properties. Our results further suggest that test suites generated by CT 3-ways, CT 4-ways, and MBT achieved similar effectiveness in detecting all functional level faults according to the achieved mutation scores. A deeper analysis, however, shows that MBT did not achieve a high mutation score for the operator LRO-I when compared with CT 3-ways and CT 4-ways and missed generating specific fault-revealing input combinations due to the random coverage criteria.

Our results also indicate that, in general, the higher MC/DC coverage corresponds to a higher mutation score. However, there are subtle differences among different techniques. For example, the difference between the achieved MC/DC coverage and mutation scores in the case of CT 2-ways and CT 3-ways is greater between other techniques.

Lastly, we argue that a mutation analysis at the FBD level alone is not sufficient to measure the fault detection effectiveness of test suites, particularly when safety critical systems are concerned, and further analysis should be conducted at the code level in industrial settings. The reason is that for similar requirements, if one FBD program is used to generate different instances of code, then a limited number of mutants can be induced as well as a fault at the FBD level can also produce multiple faults in different areas of the generated code. This has an impact on the mutation score that can be achieved per test technique. Similarly, it is also possible that a test suite may not achieve significant effectiveness in terms of fault detection rate at the code level or if different FBD programs are used to generate the code for similar requirements.

6 VALIDITY THREATS

The section provides an overview of the threats to the validity of this study along with the methods we use to counter them.

The threats related to the internal validity include the conformance of the model to system requirements and the existence of equivalent mutants. The modeling of system requirements in MBT is a manual process, and it requires a complete understanding of the system requirements and environment. One can misinterpret these requirements, impacting the conformance of the model with actual system requirements and thus impacting the test suite generated. Hence, to mitigate this factor, we have developed the model in an iterative manner and by getting continuous feedback from the testers at Alstom Transportation AB.

Similarly, in mutation testing, the existence of equivalent mutants is also a possible risk to the evaluation of test suites. We tried to eliminate this threat by spending a fair amount of time examining the alive mutants and test results manually.

The factors that can affect the reliability and external validity of this study include the particularities of the MBT model and test suites specific to the Alstom Transportation's environment, test generation tools, modeling notations, size of the subsystem, human experience, and generation algorithms. We created the model and test suites by using the requirements related to a subsystem of the TCMS developed at Alstom Transportation and it contains some particularities related to Alstom Transportation's specific testing environment, libraries, and development tools that may not be relevant to other domains. However, we have provided sufficient information on the experimentation methodology and argue that if a researcher with similar testing and modeling experience will replicate this study, similar results can be obtained. Furthermore, different test generation tools, modeling notations, and generator algorithms may also affect the results. The research into the effectiveness of test suites should therefore require more industrial case studies for creating generalizable knowledge.

The measures used for MC/DC and mutation score in this study were inspired by the literature and evaluation metrics used in industrial safety standards, i.e., ISO 26262, IEC 61508, EN 50128, and EN 50657. Moreover, the selection of mutants is done by thoroughly investigating the operators used for the development of FBD programs from the literature, their industrial applicability, and dependencies on the tools used by the Alstom Transportation for developing safety-critical programs.

7 CONCLUSION AND FUTURE WORK

This paper provides an experimental evaluation of industrial manual testing and two popular system-level automated test generation techniques, MBT and CT, in terms of fault detection effectiveness at the system level using an industrial case study. In addition, we measure the sensitivity of each test suite towards each type of fault induced by different mutant operators. Moreover, we examine the relationship between the MC/DC coverage and mutation scores at the system level. The experimental results show that the test suites achieved mutation scores within the range of 82% (CT 2-way) and 90% (CT 3-ways and CT 4-ways), whereas other techniques' mutation scores ly-

ing within this range. Thus, CT with higher interaction strength (3-ways and 4-ways) was found to be the most effective testing technique in terms of achieved mutation score, closely followed by MBT with a mutation score of 88%. This means that we found higher-interaction strength CT and MBT as most effective in detecting induced faults based on the selected mutant operators, the exception being the TRO, where none of the techniques were able to find faults based on this particular operator. MBT was found to be the least effective in detecting faults induced by the LRO-I operator, while manual testing achieved a low mutation score in detecting LRO mutants. CT 2-ways was found to be the least effective testing technique in our case. On the other hand, manual testing was also found to be efficient in terms of execution time. The results also showed that the MBT-generated test suite achieved the highest MC/DC coverage when compared with other techniques. Lastly, the analysis of mutation and MC/DC coverage scores showed a general positive relationship between both measures for all test suites. Hence, we put forward the hypothesis that the test suite achieving adequate MC/DC coverage tends to also provide a higher mutation score.

In future work, we intend to execute the generated test suites at Hardware-in-the-Loop (HiL) level and explore the optimization techniques for the MBT-generated test suite. Moreover, a thorough evaluation and statistical analysis are also warranted to analyze the mutation score and MC/DC coverage at the structural level.

ACKNOWLEDGEMENT

This work has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement Nos. 871319, 957212; from the Swedish Innovation Agency (Vinnova) through the SmartDelta project and from the ECSEL Joint Undertaking (JU) under grant agreement No 101007350.

REFERENCES

- Acree, A. T., Budd, T. A., DeMillo, R. A., Lipton, R. J., and Sayward, F. G. (1979). Mutation analysis. Technical report, Georgia Inst of Tech Atlanta School of Information And Computer Science.
- Ahmed, B. S., Enoiu, E., Afzal, W., and Zamli, K. Z. (2020). An evaluation of monte carlo-based hyper-heuristic for interaction testing of industrial embedded software applications. *Soft Computing*, 24(18):13929–13954.
- Anand, S., Burke, E. K., Chen, T. Y., Clark, J., Cohen, M. B., Grieskamp, W., Harman, M., Harrold, M. J., McMin, P., Bertolino, A., et al. (2013). An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software*, 86(8):1978–2001.
- Baker, R. and Habli, I. (2012). An empirical evaluation of mutation testing for improving the test quality of safety-critical software. *IEEE Transactions on Software Engineering*, 39(6):787–805.
- Bures, M. and Ahmed, B. S. (2017). On the effectiveness of combinatorial interaction testing: A case study. In *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 69–76. IEEE.
- Charbachi, P., Eklund, L., and Enoiu, E. (2017). Can pairwise testing perform comparably to manually hand-crafted testing carried out by industrial engineers? In *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 92–99. IEEE.
- Delgado-Pérez, P., Habli, I., Gregory, S., Alexander, R., Clark, J., and Medina-Bulo, I. (2018). Evaluation of mutation testing in a nuclear industry case study. *IEEE Transactions on Reliability*, 67(4):1406–1419.
- Enoiu, E. P., Cauevic, A., Sundmark, D., and Pettersson, P. (2016a). A controlled experiment in testing of safety-critical embedded software. In *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 1–11. IEEE.
- Enoiu, E. P., Sundmark, D., Čaušević, A., Feldt, R., and Pettersson, P. (2016b). Mutation-based test generation for plc embedded software using model checking. In *IFIP International Conference on Testing Software and Systems*, pages 155–171. Springer.
- Hamlet, R. (1994). Random testing. *Encyclopedia of software Engineering*, 2:971–978.
- Hemmati, H. (2015). How effective are code coverage criteria? In *2015 IEEE International Conference on Software Quality, Reliability and Security*, pages 151–156.
- Hierons, R. M. and Merayo, M. G. (2009). Mutation testing from probabilistic and stochastic finite state machines. *Journal of Systems and Software*, 82(11):1804–1818.
- Johnson, L. A. et al. (1998). Do-178b, software considerations in airborne systems and equipment certification. *Crosstalk, October*, 199:11–20.
- Khalsa, S. K. and Labiche, Y. (2014). An orchestrated survey of available algorithms and tools for combinatorial testing. In *2014 IEEE 25th International Symposium on Software Reliability Engineering*, pages 323–334. IEEE.
- Li, D., Hu, L., Gao, R., Wong, W. E., Kuhn, D. R., and Kacker, R. N. (2017). 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.
- Lindström, B., Offutt, J., Gonzalez-Hernandez, L., and Amdler, S. F. (2018). Identifying useful mutants to test

- time properties. In *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 69–76. IEEE.
- Maslar, M. (1996). Plc standard programming languages: Iec 1131-3. In *Conference Record of 1996 Annual Pulp and Paper Industry Technical Conference*, pages 26–31. IEEE.
- McMinn, P. (2011). 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.
- Nie, C. and Leung, H. (2011). A survey of combinatorial testing. *ACM Computing Surveys (CSUR)*, 43(2):1–29.
- Oh, Y., Yoo, J., Cha, S., and Son, H. S. (2005). Software safety analysis of function block diagrams using fault trees. *Reliability Engineering & System Safety*, 88(3):215–228.
- Petke, J., Yoo, S., Cohen, M. B., and Harman, M. (2013). Efficiency and early fault detection with lower and higher strength combinatorial interaction testing. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pages 26–36.
- Petrovic, G., Ivankovic, M., Kurtz, B., Ammann, P., and Just, R. (2018). An industrial application of mutation testing: Lessons, challenges, and research directions. In *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 47–53.
- Ramler, R., Kopetzky, T., and Platz, W. (2012). Combinatorial test design in the toasca testsuite: lessons learned and practical implications. In *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, pages 569–572. IEEE.
- Ramler, R., Wetzlmaier, T., and Klammer, C. (2017). An empirical study on the application of mutation testing for a safety-critical industrial software system. In *Proceedings of the Symposium on Applied Computing*, pages 1401–1408.
- Rouvroye, J. L. and van den Blik, E. G. (2002). Comparing safety analysis techniques. *Reliability Engineering & System Safety*, 75(3):289–294.
- Sánchez-Gordón, M., Rijal, L., and Colomo-Palacios, R. (2020). Beyond technical skills in software testing: Automated versus manual testing. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, pages 161–164.
- Schwartz, A. and Hetzel, M. (2016). 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.
- Schwartz, M. D., Mulder, J., Trent, J., and Atkins, W. D. (2010). Control system devices: Architectures and supply channels overview. *Sandia Report SAND2010-5183, Sandia National Laboratories, Albuquerque, New Mexico*, 102:103.
- Shin, D., Jee, E., and Bae, D.-H. (2012). Empirical evaluation on fbd model-based test coverage criteria using mutation analysis. In *International Conference on Model Driven Engineering Languages and Systems*, pages 465–479. Springer.
- Shin, D., Jee, E., and Bae, D.-H. (2016). Comprehensive analysis of fbd test coverage criteria using mutants. *Software & Systems Modeling*, 15(3):631–645.
- Taipale, O., Kasurinen, J., Karhu, K., and Smolander, K. (2011). Trade-off between automated and manual software testing. *International Journal of System Assurance Engineering and Management*, 2(2):114–125.
- Tian, Y., Yin, B., and Li, C. (2021). A model-based test cases generation method for spacecraft software. In *2021 8th International Conference on Dependable Systems and Their Applications (DSA)*, pages 373–382.
- Utting, M. and Legeard, B. (2010). *Practical model-based testing: a tools approach*. Elsevier.
- Vilkomir, S., Alluri, A., Kuhn, D. R., and Kacker, R. N. (2017a). Combinatorial and mc/dc coverage levels of random testing. In *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 61–68. IEEE.
- Vilkomir, S., Baptista, J., and Das, G. (2017b). Using mc/dc as a black-box testing technique. In *2017 IEEE 28th Annual Software Technology Conference (STC)*, pages 1–7. IEEE.
- Wagner, M., Kleine, K., Simos, D. E., Kuhn, R., and Kacker, R. (2020). Cagen: A fast combinatorial test generation tool with support for constraints and higher-index arrays. In *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 191–200. IEEE.
- Yao, X., Harman, M., and Jia, Y. (2014). A study of equivalent and stubborn mutation operators using human analysis of equivalence. In *Proceedings of the 36th international conference on software engineering*, pages 919–930.
- Zafar, M. N., Afzal, W., and Enoiu, E. (2022). Evaluating system-level test generation for industrial software: A comparison between manual, combinatorial and model-based testing. In *Proceedings of the 3rd ACM/IEEE International Conference on Automation of Software Test, AST '22*, page 148–159, New York, NY, USA. Association for Computing Machinery.
- Zafar, M. N., Afzal, W., Enoiu, E. P., Stratis, A., Arrieta, A., and Sagardui, G. (2021a). Model-based testing in practice: An industrial case study using graphwalker. In *Innovations in Software Engineering Conference 2021*.
- Zafar, M. N., Afzal, W., Enoiu, E. P., Stratis, A., and Sellin, O. (2021b). A model-based test script generation framework for embedded software. In *The 17th Workshop on Advances in Model Based Testing*.