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

Muhammad Nouman Zafar
muhammad.nouman.zafar@mdh.se
Mälardalen University
Sweden

Wasif Afzal
wasif.afzal@mdh.se
Mälardalen University
Sweden

Eduard Enoiu
eduard.enoiu@mdh.se
Mälardalen University
Sweden

ABSTRACT

Adequate testing of safety-critical systems is vital to ensure correct functional and non-functional operations. Previous research has shown that testing 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.

AST '22, May 17–18, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9286-0/22/05.
https://doi.org/10.1145/3524481.3527235

CCS CONCEPTS

• **Computer systems organization** → **Embedded software**; • **Software and its engineering** → **Software verification and validation**.

KEYWORDS

Test Generation, Test Coverage, System-level Tests, Safety Critical Systems

ACM Reference Format:
Muhammad Nouman Zafar, Wasif Afzal, and Eduard Enoiu. 2022. Evaluating System-Level Test Generation for Industrial Software: A Comparison between Manual, Combinatorial and Model-Based Testing. In *IEEE/ACM 3rd International Conference on Automation of Software Test (AST '22), May 17–18, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3524481.3527235

1 INTRODUCTION

The testing of safety-critical systems help avoid abrupt behaviors that can cause catastrophic events. To achieve efficiency and effectiveness in testing, automated testing tools have been proposed in the late 1970s on-wards, with a surge after 2000s. They are based on generation of tests using techniques such as combinatorial testing (CT), equivalence partitioning (EP) etc. and help fulfill coverage criteria such as requirement coverage and Modified Condition/Decision Coverage (MC/DC).

The challenging part in testing is the generation of cost-efficient and high-quality tests, which can provide better specification as well as code coverage, within respectable time. Some studies (e.g., [29], [34], [35]) have also integrated multiple testing techniques and coverage criteria for test suite optimization. For safety-critical vehicles, ISO 26262[1] recommends the Modified Condition/Decision Coverage (MC/DC) adequate test suites at all of the Safety Integrity Levels where complex Boolean expressions are involved. There is also some evidence to suggest the effectiveness of MC/DC in terms of fault detection for complex systems [12] [46]. However, the generation of MC/DC adequate tests is an expensive process as well as requires a lot of effort [17].

Multiple researchers have been exploring various techniques to reduce the complexity and cost of test generation while achieving higher MC/DC. One of such proposed techniques is symbolic execution, which uses the constraint solving algorithm to generate MC/DC adequate tests [6] [16] [36]. But symbolic execution has the drawback of path explosion problem, due to which constraints solving and execution analysis require a lot of computational power to generate test cases [2]. Another study by Li et al. [29] showed that

[1] https://www.iso.org/standard/68383.html

higher MC/DC coverage can be achieved by increasing the strength of combinatorial testing. Similarly, model-based testing (MBT) has been used to generate MC/DC adequate test suites [13] [38]. In addition to the above mentioned automated test techniques, manual testing is still a prevalent technique in industry [41]. Manual testing capitalizes on both technical and non-technical skills that are required for effective testing [1] [39]. Hence, a comparison of industrial manual testing with automated test techniques, in terms of MC/DC coverage in the context of safety-critical systems, can help industry invest resources in suitable techniques.

There exists several state-of-the-art studies on comparative analysis of automated testing techniques such as random testing [44], combinatorial [29] and concolic testing [27] etc., such studies are dependent on source code analysis and it is not evident if the results would stand at higher levels of testing where source code access is difficult or even not possible. Thus in this paper, we investigate MC/DC adequacy of two automated testing techniques (CT and MBT) and industrial manual testing at *system-level*, along with measuring requirement coverage and performance efficiency in terms of time. We also perform an assessment of the differences and overlaps between the test suites generated by automated testing and manual industrial testing to better demonstrate the potential gains and trade-offs among them, with the goal that an optimal testing strategy can be reached.The comparative evaluation is done on an industrial safety-critical system on-board the passenger trains that controls critical train functions. The system is called as the Train Control Management System (TCMS) and is developed at Alstom Transport AB in Sweden.

The results of this paper show that:

- The test suite generated by MBT provided higher MC/DC coverage than the test suite generated by CT and written manually.
- The test suite generated by the 2-ways strategy of CT provided only partial requirement coverage.
- MBT-generated test suite acted as a super set of test cases by containing approximately 71% of similar test cases, on average, produced by other selected techniques.
- MBT-generated unique test cases were highly relevant to MC/DC coverage whereby they will complement manual testing to increase MC/DC coverage.
- The overall cost of test suite generation and execution of CT was significantly lower as compared to MBT and manual testing, but with an additional cost parameter of manual identification of expected outputs.

The rest of the paper is organised as follows: Section 2 deals with background and related work, Section 3 provides an overview of the experimental setup including a description of the System under test (SUT), research questions, a description of the evaluation metrics and assumptions, Section 4 describes the results, Section 5 discusses the results, Section 6 presents the validity threats whereas conclusions and future work are presented in Section 7.

2 BACKGROUND AND RELATED WORK

This section presents a background on test generation techniques and coverage criteria used in this paper along with a summary of related work.

2.1 Test Generation Techniques

A primary purpose of the test generation techniques is to develop high-quality test suites containing finite and relevant test cases to detect faults, efficiently. Anand et. al [2] specified and reviewed various test generation techniques including specification-based, goal-oriented, code-based, random and model-based testing. When it comes to CT and MBT, multiple studies, e.g., [4], [5], [13], [18], [21], [26] report their effectiveness and efficiency at industrial, system level testing.

CT generates test suites based on input combinations and to test functional and non-functional requriements of a SUT. The behaviour of a software system depends on multiple input parameters but to validate a system using all combinations of these parameters is an exhaustive and complicated process. CT generates test suites containing minimum but relevant combinations of input parameters known as a covering array [9] based on t-ways strategy where t is the interaction strength. t-ways strategy provides various interaction possibilities based on heuristic search between the input parameters and value of *t*. It generates unique sequences of test data, which ensures that each combination based on t-ways (2-ways, 3-ways, 4 ways and so on) parameter interaction is covered at least once. For instance, in case of pairwise (2-ways) strategy, the test cases contain combinations covering all the pairs of input parameters at least once.

In this study, we have selected CAgen[2] as a combinatorial testing tool due to its comparatively high performance than other state-of-the-art combinatorial testing tools [45]. Moreover, it is an open-source tool and offers three different heuristic algorithms (i.e. FIPOG, FIPOG-F, FIPOG-F2) to generate t-ways test cases. It has flexible availability in two versions, an online web GUI version and an offline command line version. In this study, we have used the online version of CAgen for test case generation with FIPOG heuristic algorithm, test case redundancy value '1' and randomization of don't care values.

MBT [42] is one of the advanced techniques for automated test generation. In MBT, test artefacts are generated using an explicit model representing the environmental, functional and non-functional behaviour of the SUT based on a certain modelling language and notation such as Unified Modelling Language (UML), finite state machine (FSM), Finite Automata etc. For example, a FSM model of a SUT contains nodes representing states of the system, edges representing the transitions between multiple states based on specified actions and guard conditions. The guard conditions can be the Boolean constraints depicting the expected behavior of the SUT. After the creation of the FSM model, abstract test cases can be derived using multiple MBT defined coverage criteria (edge, node and requirement etc.) and generator algorithms (random, quick random and A-star etc.). The abstract test cases are then transformed into concrete or executable test cases to produce test verdicts.

Multiple MBT tools are available [30] for the generation of test artefacts based on different modelling languages/notations, coverage criteria and generator algorithms. However, in this paper, we have used TIGER (Model-Based Test scrIpt GenEration fRamework) [49], developed using the open-source FSM-based

[2] https://matris.sba-research.org/tools/cagen/#/about

MBT tool, GraphWalker[3] (GW). TIGER is capable of generating system-level test scripts, which contain all the implementation details about test execution framework used at Alstom Transport AB, Sweden. It takes the FSM model as an input and traverses through its nodes and edges based on a given coverage criterion (e.g. egde_coverage, node_coverage etc.) and generator algorithm (e.g. random, quick_random etc.) to generate test sequences.

2.2 Test Coverage

The test coverage criterion measures the number of items (e.g., requirements, code elements) covered by a test suite. There exist multiple coverage criteria at design and implementation levels such as requirement coverage, statement coverage, decision coverage, condition coverage etc. However, requirement and MC/DC adequate test suites have been recommended by various standards i.e. EN 50128, EN 50657 [7] and ISO26262 to verify and validate a safety critical system. Moreover, Arefin et al.. [21] showed that, for a safety critical system, MC/DC can be used at system-level to evaluate the effectiveness of a testing technique.

Requirement coverage is one of the basic and well-known black box testing coverage criterion, which is used to measure the extent of critical and domain requirements covered by a test suite at least once. It is also used to discover the existence of documented and undocumented requirements in the implementation of a system software and to determine the number of test cases required to cover all the system requirements [23].

Modified Condition/Decision Coverage (MC/DC) [8] is one of the strongest coverage criterion to measure the structural coverage of a system, which subsumes branch coverage as well as statement coverage. It ensures the coverage of every logical predicate in the code along with the coverage of each condition in a decision. Moreover, it also ensures that every condition has an independent effect on the outcome of a decision. Hence, a test suite is said to be MC/DC adequate if it fulfills each of the following conditions [20]:

- Every entry and exit point of a program must be invoked at least once.
- All possible outcomes of a condition should be taken at least once.
- All possible outcomes of a decision should be taken at least once.
- Each condition must have an independent effect on the outcome of a decision.

2.3 Related Work

Manual testing is still considered as an equally prevalent testing technique in industrial settings and has been used in different domains [32]. There are several state-of-the-art studies (e.g. [3], [4], [10], [14], [15], [21], [29], [31], [34], [35], [40]), which have explored, compared and evaluated CT, MBT and manual testing techniques. We have summarized these studies in three categories as given below.

2.3.1 Manual Vs MBT. Enoiu et. al [14] have provided a comparative analysis between manual industrial practices and automated model-based testing technique using TCMS as an industrial case study in terms of decision coverage, fault detection effectiveness and test generation efficiency. The results showed that the MBT is efficient for test suite generation and generated test suites are equally prevalent as a manually created test suite for decision coverage. However, the manually created test suites provided better mutation scores, consequently proved to be more effective in terms of fault detection. Christoph et. al [40] also performed a similar empirical comparative analysis between manual and model-based testing technique using a web-based system. The analysis showed MBT as an efficient technique in terms of test suite generation and the generated test suite proved to be more effective in terms of fault detection by providing a 60% higher severity score than manual created test suite. Another comparative analysis between model based and traditional testing strategies has been performed by Arthur et. al [31]. They have done an empirical evaluation to analyse the performance between the tests developed using manual ad hoc testing tool and MBT tool in terms of time, precision, relative recall and effectiveness. The results showed that manual tests are more effective for detecting system logic defects whereas MBT generated tests are effective for detecting specification level defects of a complex system.

2.3.2 Manual Vs CT. CT has been introduced as a technique to measure the quality of test suite in [25] and Miraldi et. al [15] have measured the combinatorial coverage of manual test suite using an industrial case study. They showed that the combinatorial coverage of a manual test suite decreases with increase in interaction strength of CT. Moreover, missing combinations of CT can provide useful information to improve the manually created test cases. The effectiveness of CT interaction strengths has also been evaluated using model-based mutation testing in [10]. The experimental evaluation illustrated that higher interaction strength of CT is more likely to have a high fault detection rate. Josip et. al [4] have proposed a novel approach based on combinatorial and model-based testing for the generation of test suites and compared the proposed approach with a state-of-the-art manual testing tool in terms of fault exploitation. The results depict that the automated tool has provided better fault detection rate than manual testing tool.

2.3.3 Proposed Approaches with CT and MBT. In addition to comparing the manual testing with MBT and CT, there also exist multiple studies which has proposed hybrid approaches combining the aforementioned techniques to increase the test generation efficiency and fault detection effectiveness of test suites.

A hybrid test suite generation approach based on CT and MBT to validate event-based systems has been proposed in [3]. The authors have used an FSM model represented by automata and generated the combinatorial test sequences based on constraints. They have also compared the generated test suite with test suite generated by t-ways covering arrays to evaluate the performance of the approach and showed its effectiveness in terms of valid test sequence generation and coverage. Another approach to generate an effective test suite with a combination of CT and MBT has been proposed in [34]. They also provided an empirical evaluation of the generated test suite based on number of test cases, code coverage and fault detection effectiveness. The proposed approach found to be effective in terms of fault detection and code coverage. A tool chain using pairwise and model-based testing techniques has been

[3] http://graphwalker.github.io/

presented and evaluated in [35]. The proposed tool chain generated a reduced amount of test sequences and reusable test artefacts.

2.3.4 Research Gap. It is important to mention that the aforementioned studies have performed the comparison and evaluation in terms of fault detection effectiveness, efficiency and structural coverage such as decision coverage and statement coverage. There also exist various studies which have generated MC/DC adequate test suites by proposing multiple techniques (e.g. [21], [29]). However, comparison between the test suites developed by MBT, CT and manual testing practices in terms of differences and overlaps has not been found to the best of our knowledge. Moreover, it is also interesting to investigate and evaluate these techniques in terms of MC/DC and requirement coverage at higher testing levels where the source code is difficult to access or inaccessible.

3 METHODOLOGY

Figure 1 represents an overview of the methodology that we designed and followed for our study. The sequence of steps in our methodology are represented with numbers in Figure 1 and described below:

- (1.1) & (1.2) the analysis of requirements specification and test specification using an industrial case study of TCMS
- (2.1) & (2.2) identifying the parameters involved in defining the behaviour of the SUT and creating a model of the system in JSON format using GW
- (3.1), (3.2) & (3.3) generation of manual test suite by testers at Alstom Transport AB and automated test suites by CAgen and TIGER (Section 3.3)
- (4) evaluating the test suites based on requirement and MC/DC coverage metrics (Section 3.4)
- (5) measuring the efficiency of test suites based on a cost model (Section 3.5)
- (6) assessing the differences and overlapping between the test suites (Section 3.6).

The motivation for selected CT and MBT tools (i.e. CAgen and TIGER) has been presented in Section 2.1.

3.1 Research Questions

The aim of our study is to answer the following research questions:

- **RQ1:** How does the use of MBT, CT and manual testing impact the requirement and MC/DC coverage?
- **RQ2:** What is the cost of generating and executing test cases using MBT, CT and manual testing, while achieving MC/DC coverage?
- **RQ3:** What are the differences and overlaps between the test suites developed using MBT, CT and manual testing techniques?

3.2 Description of SUT

We have selected a subsystem of the Train Control Managment System (TCMS), from an on-going project MOVIA [4] family of vehicle products developed at Alstom Transport AB as an industrial case study. TCMS is the central part of a distributed control system, which is used to control and manage all the functional operations of the train. It uses an open standard IP-technology to interact and control different subsystems of a modern train such as doors, ventilation and brakes. TCMS consists of multiple devices which are connected via various networks such as Ethernet Consist Network (ECN) and Multi-function Vehicle Bus (MVB) [48]. Some of these devices are also connected to a Centralized Traffic Control (CTC) which deals with the routing decision of a train. However, Modular Input/Output-Safe (MIOS) and Central Control Unit-Safe (CCUS) are the devices which deal with safety related functions of a train. MIOS is responsible for handling the input/output analog signals whereas CCUS contains the logical components to execute the safety critical functions of a train. In discussion with Alstom Transport AB, we have selected the requirements of a fire detection subsystem. Figure 2 represents an architecture of the fire detection system used by TCMS.The fire detection system of TCMS uses two instances of Fire Detection Control Units (FDCUs) to detect fire based on their current states i.e., Master or Slave. A FDCU is considered as 'Master' if two of its signals hold the value as true. Both FDCUs communicate with smoke and fire sensors to receive signals indicating two types of fire i.e., internal and external. The TCMS MIOS device receives these signals from both FDCUs and communicates with CCUS via the MVB network. The CCUS computes the logic based on system requirements and responds with an output signal. The MIOS receives the corresponding output signal from the CCUS and indicates the type of fire on the driver's desk.

Figure 3 shows the model of the fire detection subsystem based on a FSM. Figures 3(a) and 3(b) represent the FDCUs while diagram (c) represents the TCMS as black box. The nodes depict the states of the SUT and edges show the signal transitions between these states. `FDCU1` is the initial as well as shared state of the model, whereas `FDCU1Signal`, `FDCU2`, `TCMSisActive` and `FDCUsFireSignals` represent the shared states. All shared states are used by GW to traverse through different diagrams of the model. It also contains actions and guard conditions according to the expected behaviour of the SUT. For further details regarding this model, the interested reader is referred to our earlier publications, [47], [48], [49].

3.3 Test Suite Creation

We took manual test suite created for the selected requirements by the testers at Alstom Transport AB according to EN 50128 and EN 50657 safety standards and regulations. The test suite consists of a set of test steps specifying test inputs, expected outputs and timing constraints in natural language based on specification-based testing. The design process of the test suite ensures that each requirement of the system has been covered and executed. After the designing of test cases, test scripts are created manually, executed on Software-in-the-Loop (SiL) level by a testing framework and a test verdict is generated.

For the generation of test suites for CT and MBT, we have utilized CAgen and TIGER tools respectively. Particularly, we have considered all the signals used by the SUT for communication between FDCUs, sensors, and TCMS as parameters and generated 2-ways, 3-ways, and 4-ways test suites, in the form of decision tables, through the CAgen tool. Furthermore, TIGER generated the test suite based on FSM model as shown in Figure 3 using 100% edge coverage and through random algorithm. It is important to

[4] https://www.railway-technology.com/projects/bombardier-movia-metro-cars/

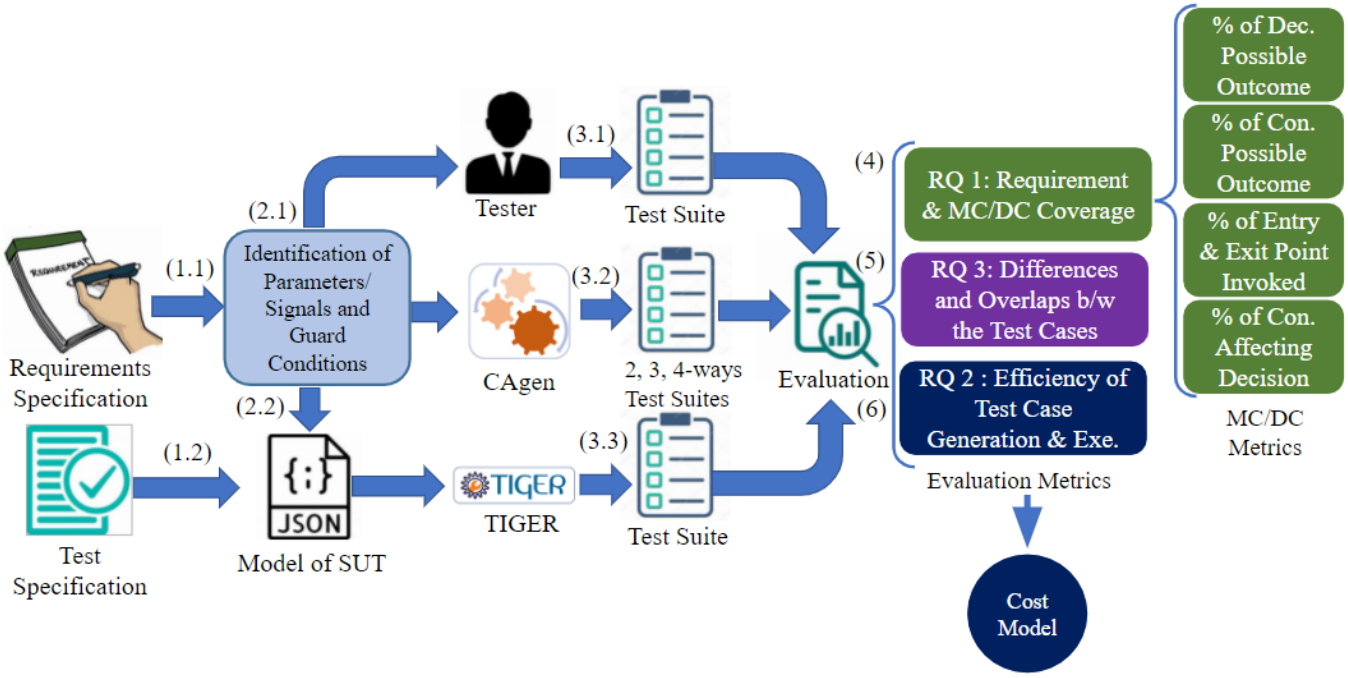


Figure 1: An overview of the experimental methodology

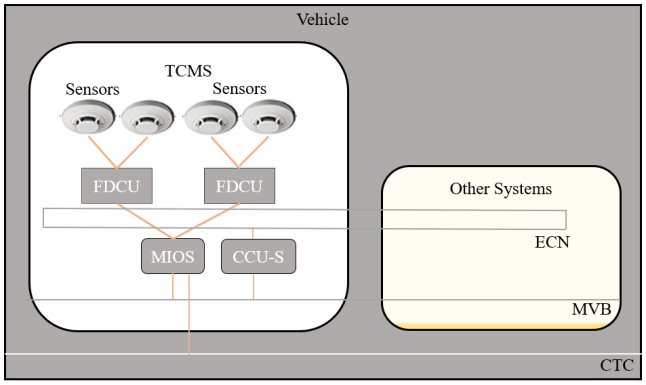


Figure 2: Architecture illustration of fire detection subsystem of TCMS

mention that test scripts are developed manually in case of CT but are generated automatically in TIGER for MBT. The reason behind limiting CT at 2, 3 and 4-ways interaction strength is the combinatorial explosion of test cases generated for complex systems with higher strength [37]. Similarly, the random generation algorithm and edge coverage criterion of GW provides adequate model and requirement coverage, which is an important metric for our industrial partners.

3.4 RQ1: Coverage Assessment

A coverage metric is used to analyse the quality of a test suite and measures the degree to which the code has been exercised by a test suite. The testing of safety-critical subsystems of TCMS at Alstom Transport AB need to follow EN 50128 and EN 50657 standards. These standards require that the code related to each requirement should be executed by the test suite. Hence, in this study, we have defined two metrics, i.e., requirement coverage and MC/DC to analyse the design and structural coverage provided by a test suite.

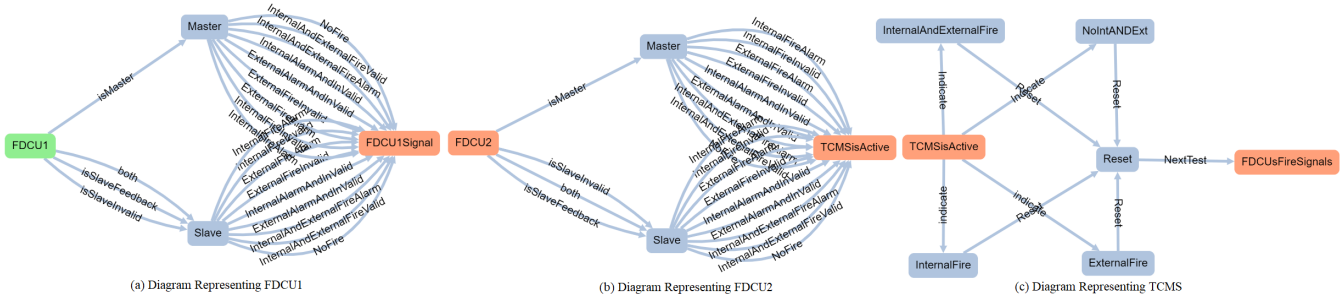
The requirement coverage requires that requirements of a system are exercised by test cases at least once [43]. Hence, we identified the total number of requirements specified in requirement specification and analysed the requirements covered by each test suite to measure the requirement coverage using the following formula:

$$RequirementCoverage(RC)\% = (NRC/TNR)x100$$

Where NRC represents number of requirements covered by test cases and TNR depicts total nnumber of requirements.

As the requirement coverage does not ensure the coverage of logical conditions and decisions of a system from the implementation perspective, we have also used MC/DC to examine the structural coverage of the test suites. We have defined the MC/DC metric according to the coverage points defined for a MC/DC analysis tool in [20]. Moreover, we have also analysed each condition of MC/DC metric to investigate the least dominant condition affected by each technique to generate MC/DC adequate test suites. We used the following formulas to determine the MC/DC of test suites in percentage:

- Condition's possible outcomes (C) % = (No. of conditions having all possible outcomes/Total no. of conditions) x 100
- Decision's possible outcomes (D) % = (No. of decision having all possible outcomes/Total no. of decisions) x 100



(a) Diagram Representing FDCU1 (b) Diagram Representing FDCU2 (c) Diagram Representing TCMS

Figure 3: An FSM model representing the fire detection subsystem of the TCMS

- Conditions affecting decisions independently (AC) % = (No. of conditions affected the decisions / Total no. of independently effecting conditions) x 100
- Entry and exit point invoked (E) % = (Sum of no. of entry and exit points invoked/Sum of total no. of entry and exit points) x 100

Hence, we have calculated the overall MC/DC of a test suite using the formula below.

$$MC/DC\% = (C + D + AC + E)/4$$

3.5 RQ2: Efficiency Evaluation Criteria

The efficiency of testing techniques can be measured by analysing the cost/time benefit of each technique based on direct and indirect costs [28]. The direct costs are the expenditures that are directly associated with testing techniques such as time for test creation and execution of test cases. The indirect costs are the expenditures that are indirectly associated with the testing techniques such as maintenance time, development of testing tools etc. Hence, to analyse the efficiency of a testing technique, we have developed an economic cost model based on direct costs and have neglected the indirect costs as the effect of indirect costs diminishes over time, for example, development of a tool is only an one-time effort. Therefore, in this model, we have considered the following parameters based on variable cost factors and time required to complete the activities defined in [23] and [33] as well as the actual activities adopted by testers at Alstom Transport AB.

The cost of testing activities that we have considered in our efficiency evaluation metrics for automated and manual testing include analysis of requirements specification (C_R), test suite development (C_{TS}) and test execution (C_E). The requirements analysis involves understanding the behaviour of a system by thoroughly investigating functional (C_{Rf}) and non-functional (C_{Rnf}) requirements as well as reviewing (C_{Rr}) these requirements to determine unfeasible requirements before the start of the development process. After requirements analysis, test cases are designed followed by the development of test scripts and identifying concrete values (e.g. technical signal names) (C_s). The test scripts containing test steps (C_{Ets}) are then executed along with pre and post conditions (C_{Epre}, C_{Epost}) to generate test verdicts. However, in case of MBT in our context, a model is created (C_M) by identifying the states, transitions (C_{Mcst}), guard conditions (C_{Mcg}) and test scenarios (C_{Mct}) based on requirements analysis. Moreover, the test suite (C_{Gts}) is generated automatically by creating an XML file containing information about the signals (C_{Gf}) (e.g. technical signal names, primary signal names and data type etc.) and ensuring the correctness and conformance of a model (C_{Mv}) with system requirements. On the other hand, in case of manual testing, a tester designs test cases (C_{Tc}) and scripts (C_{Tsd}) manually by specifying and developing system libraries (C_{sl}).

3.5.1 *Parameters for cost calculation of MBT.*

- C_R = Cost for Requirements Specification Analysis (C_{Rf} + C_{Rnf} + C_{Rr})
  - C_{Rf} = Cost to analyze functional Requirements
  - C_{Rnf} = Cost to analyze non-function Requirements
  - C_{Rr} = Cost to review requirements
- C_M = Cost for modelling the SUT (C_{Mc} + C_{Mv})
  - C_{Mc} = Cost to create and modify model (C_{Mcst} + C_{Mcg} + C_{Mct})
    - C_{Mcst} = Cost to identify states and transitions
    - C_{Mcg} = Cost to identify guard conditions
    - C_{Mct}= Cost to identify test scenarios
  - C_{Mv} = Cost to validate the correctness of model
- C_{TS} = Cost to generate MBT test cases and test scripts (C_{Gf} + C_{Gts})
  - C_{Gf} = Cost to prepare XML file containing logical and technical signal names
  - C_{Gts} = Cost to generate MBT abstract test cases and concrete test scripts
- C_E = Cost to execute MBT test scripts
  - C_{Epre} = Cost to execute pre-conditions
  - C_{Ets} = Cost to execute test steps
  - C_{Epost}= Cost to execute post-conditions

Based on the above defined parameters, the cost for MBT (C_{MBT}) can be summed below:

$$C_{MT} = C_R + C_M + C_{TS} + C_E$$

3.5.2 *Parameters for cost calculation of manual testing.*

- C_R = Cost for Requirement Specification Analysis (C_{Rf} + C_{Rnf} + C_{Rr})
  - C_{Rf} = Cost to analyze functional Requirements
  - C_{Rnf} = Cost to analyze non-function Requirements
  - C_{Rr} = Cost to review requirements
- C_{TS} = Cost for the development of test suite (C_s + C_{Tc} + C_{Ts})

  - C_s = Cost to Identify technical signals to use in test design
  - C_{Tc} = Cost for writing/designing test specification – test cases
  - C_{Ts} = Cost for writing concrete test scripts (C_{Tsl} + C_{Tsd})
    * C_{Tsl} = Cost for specifying and developing library functions
    * C_{Tsd} = Cost to develop test scripts
- C_E = Cost to execute test scripts
  - C_{Epre} = Cost to execute pre-conditions
  - C_{Ets} = Cost to execute test steps
  - C_{Epost}= Cost to execute post-conditions

Based on the above defined parameters, the total cost for manual testing (C_{MT}) can be summed below:

$$C_{MT} = C_R + C_{TS} + C_E$$

3.5.3 *Parameters for cost calculation of CT.*

- The testing activities required for CT include all the activities similar to manual testing, just that test case development is automated in CT in our case. Hence, we have used the aforementioned parameters of manual testing to determine the total cost for CT (C_{CT}) too.

$$C_{CT} = C_R + C_{TS} + C_E$$

3.6 RQ3: Differences and Overlaps Assessment

To measure the differences and overlaps between the test suites produced by each testing technique, we investigated the similar test sets and created a Venn diagram to show the distribution and intersection of different test sets between test suites. A Venn diagram shows the measure of union for sets using the measure of intersections between them. Hence, the illustration of relationships between the elements of test suites in Venn diagram is done using Meta-Chart[5] which is an online data visualization tool. We provided the total number of elements and intersections between different combinations of test suites to create the Venn diagram.

3.7 Assumptions

In evaluation metrics mentioned above, we have made some explicit assumptions:

- In the coverage metric based on MC/DC, as we do not have access to the source code; we assume that every entry and exit point of a program is invoked when a condition for a decision is evaluated as true.
- In the cost model, we estimated the execution time only for the passed test scripts.
- For the differences and overlaps between test suites, there are multiple combinations in a test case which can be considered as 'don't care' (having no effect on the decision), so we considered only those test cases similar across the test suites and redundant within the test suite which contains similar combinations of inputs affecting the decision of a system.

4 RESULTS

In this section, we provide a quantitative analysis of the data to answer the research questions.

[5] https://www.meta-chart.com/venn#/data

4.1 RQ1: Requirement and MC/DC Coverage

The requirement and MC/DC coverage of test suites generated by the three different techniques (MBT, manual and CT) is shown in Figure 4 whereas Figure 5 represents the breakdown of MC/DC coverage according to the metric parameters.

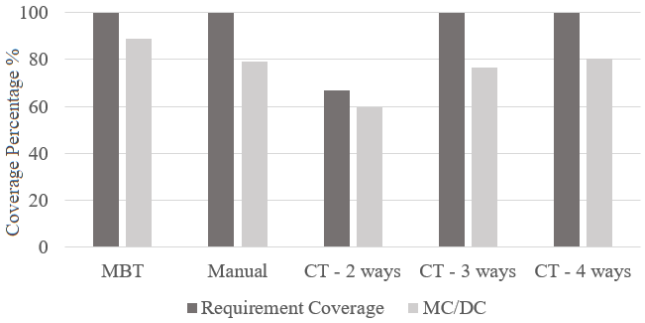


Figure 4: Requirement and MC/DC coverage of test suites

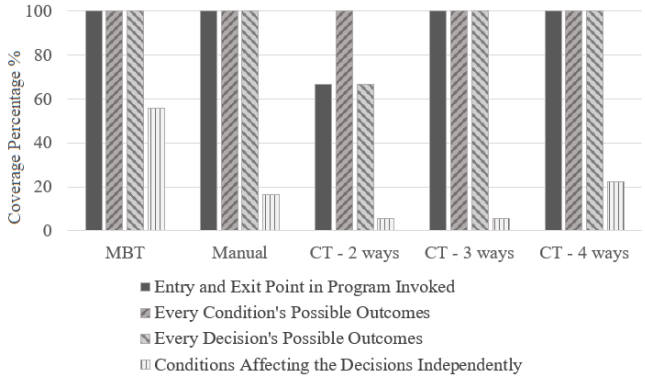


Figure 5: MC/DC coverage of test suites according to selected parameters

The results showed that all the test suites contained test cases covering each requirement of the system at least once except the test suite generated by 2-ways combinatorial test strategy that provided 66% requirement coverage. However, the MBT-generated test suite is the most MC/DC adequate by providing 88% MC/DC coverage as shown in Figure 4. Test suites generated using 2-ways, 3-ways, 4-ways combinatorial test strategy and by manual testing provided 59%, 76%, 80% and 79% MC/DC coverage, respectively.

The analysis of test suites in terms of MC/DC also showed the effect of each technique on the conditions of MC/DC as shown in Figure 5. The results indicated that each technique affected the generation of such test cases which can be used to verify the 'effect of a condition independently on the outcome of a decision'. Consequently, the MBT-generated test suite covered 55% of conditions to verify the independent effects on a decisions, whereas test suites generated using 2-ways, 3-ways, 4-ways combinatorial test strategy, and created manually covered 5%, 5%, 22% and 16% of such conditions, respectively. However, in case of CT test strategies, two

other conditions were affected to a larger extent with an increase in the strength of CT (i.e. for 3- and 4-ways). These two conditions are 'invocation of each entry and exit point in a program' and 'every decision possible outcome'.

Moreover, we have also analysed the MBT and CT test generation process to assess the impact of achieving the higher MC/DC on the number of test cases. As MBT can generate different numbers of test cases due to random walks, so we have generated multiple test suites using TIGER and selected three based on the number of test cases (min, max, median) and performed a comparative analysis with 2-, 3-, and 4-ways testing strategy of CT. The results showed that random walks of MBT had an intermittent effect on the number of test cases and a minimal effect on the MC/DC coverage In test suites as shown in Table 1, MBT generated 150, 245 and 145 test cases, including the redundant test cases, and provided 88%, 90% and 87% of MC/DC coverage respectively. In cases of CT, the number of test cases increase exponentially while achieving the higher MC/DC by increasing the strength of CT.

Table 1: MC/DC coverage of MBT and CT

Test technique	No. of test cases	MC/DC %
MBT (Median)	150	88.88
MBT (Max)	245	90.27
MBT (Min)	145	87.5
CT 2-ways	10	59.71
CT 3-ways	22	76.38
CT 4-ways	50	80.55

The analysis of test suites in terms of MC/DC and requirement coverage showed that MBT and manually developed test suites were more MC/DC and requirement coverage adequate and contained complete test cases (we designate a test case as complete having input(s), expected output(s) and timing constraint(s)). The completeness of test cases is dependent on the availability of the required information about the system, thus one can argue that the test engineer and the model of the SUT utilized all the relevant information such as the behaviour of the system (functional and non-functional requirements), test scenarios known to the tester based on domain knowledge and experience, and information regarding system environment to create test suites. Our results also showed that adequacy of CT-generated test suite increased with an increase of interaction strength. Moreover, it contained no information about the expected behaviour of the system and test environment. Hence, tester's assistance was required to specify the expected outputs in order to complete the test suites (this cost is captured as part of the parameter, C_{Tsd} = Cost to develop test scripts, in Section 3.5).

4.2 RQ2: Efficiency of Test Suites

Efficiency of test suites is generally calculated based on three factors i.e., time required for requirements analysis, time required for creation of a test suite and time required for execution of it. However, for simplicity, we have not reported the time for the analysis of requirements specification in the results as there was no significant difference between the time spent on the analysis phase by each technique. Moreover, we have divided the test suite development time according to the activities required by each testing technique as shown in Table 2. We have also reported the accumulated time of different activities required for modelling and execution phases as provided in Section 3.5. It is also important to mention here that the time required by a tester for adding expected outputs to complete the CT-generated test cases is also included in test script development time of CT.

The results showed that the development of a test suite using MBT was less efficient than CT due to additional activities i.e., modelling of the SUT and verification of the model's conformance with system requirements. The identification of signals in case of MBT also had a significant effect on the development time as it also involved the creation of XML file to generate executable test scripts. Whereas regardless of manual test script development in CT, CT required a significantly less test suite development time as compared to MBT and Manual testing.

The analysis also showed that the number of generated test cases in a test suite had a significant affect on the execution time. Consequently, the test suite generated using 2-ways test strategy required minimum execution time. The execution times required by the test suites generated using 3-ways, 4-ways testing strategies and MBT were greater than the manual test suite due to the additional number of test cases (more specifically, the test steps). However, based on the total time of all activities required by each testing technique, CT resulted in being the most efficient, MBT stood second and manual testing came out to be the least efficient technique.

4.3 RQ3: Differences and Overlaps Between the Test Generation Techniques

To measure the differences and overlaps between the test suites, we have identified similar test cases in each test suite generated from different techniques and created a subsequent Venn diagram as shown in Figure 6. However, we have removed the redundant test cases within each test suite to provide one-to-one mapping between the similar test cases generated by each technique. Table 3 presents an illustration of the total number of non-redundant test cases within the test suite, unique test cases across the test suites, and percentage of similar test cases across the test suites on average, whereas Tables 4 and 5 show the differences and number of test cases overlapping between the pair of each test suite, respectively.

The analysis of Figure 6 and Table 5 showed that the MBT-generated test suite contained the highest numbers of test cases generated by other techniques (approx. 71% of test cases on average as shown in Column 4 of Table 3). The test suites developed by CT using 4-ways, 3-ways, 2-ways, and manual testing contained approx. 53%, 44%, 25% and 54% of similar test cases on average, respectively. MBT and 2-ways had the greatest overlap (MBT suite contained almost 90% (7 out of 8) of test cases generated using 2-ways). MBT-generated test cases had the second greatest overlap with manual test cases whereby MBT suite contained 76% of test cases from manual (13 out of 17). The analysis of test suites also showed that the test suite generated by MBT was more similar, and hence complete, to specification-based manual testing as it contained constraints (i.e. time) and test inputs along with expected outputs, which can be used to determine requirement coverage

Table 2: Efficiency measurements (in seconds) of test suites created by each technique. N/A is short for not applicable.

Techniques	No. of test cases	Development time of test cases and scripts (C_M), (C_{TS})					Execution time (C_E)	Total time
		Modelling of SUT	Verification of model	Signal identification	Test case development	Test script development		
Manual	17	N/A	N/A	2400	15600	9600	300	27900
MBT	150	16800	1800	2700	5	2	600	20107
CT 2-ways	10	N/A	N/A	2400	2	7200	240	9842
CT 3-ways	22	N/A	N/A	2400	2	10200	360	12962
CT 4-ways	50	N/A	N/A	2400	2	13800	420	16622

and traceability. However, the generation of MBT-based test suite is dependent on the conformance of the model representing the SUT. CT-generated test suites did not contain expected outputs but provided more test scenarios that can be used to validate the model as well as to complement the manual testing of the SUT.

It can be analyzed from Figure 6 and Column 3 of Table 3 that MBT generated most of the unique test cases. Particularly, it is interesting to see the impact of unique test cases in each test suite on MC/DC and requirement coverage. Hence, for experimentation purposes, we discarded all the unique test cases from each test suite and measured their impact. Subsequently, results indicated that MC/DC coverage reduced from 88% to 84% in case of MBT. However, no change was observed in requirement coverage of the MBT-generated test suite. In case of other testing techniques, no major effect on MC/DC and requirement coverage was reported. We conclude that MBT-generated unique test cases are also highly relevant for covering MC/DC in a safety critical system as compared to the unique test cases generated by other approaches. Moreover, addition of these unique test cases will improve the effectiveness of manually created test suite in terms of MC/DC coverage.

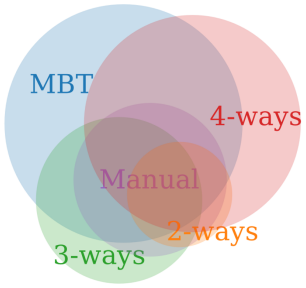


Figure 6: Venn Diagram representing the differences and overlaps between the test suites

Table 3: Number of non-redundant test cases within each test suite and unique test cases across the test suites

Technique	No. of non-redundant test cases within each test suite	No. of unique test cases across the test suites	Percentage of similar test cases across the test suites on average %
MBT	39	9	71
Manual	17	2	54
CT 2-ways	8	1	25
CT 3-ways	20	2	44
CT 4-ways	34	3	53

Table 4: Number of different test cases in each pair of test suite

Technique	MBT	Manual	CT 2-ways	CT 3-ways	CT 4-ways
MBT	N/A	26	32	28	16
Manual	4	N/A	10	7	4
CT 2-ways	1	1	N/A	3	2
CT 3-ways	9	10	15	N/A	12
CT 4-ways	11	21	28	26	N/A

Table 5: Number of overlapped test cases in each pair of test suite

Technique	Manual	CT 2-ways	CT 3-ways	CT 4-ways
MBT	13	7	11	23
Manual	N/A	7	10	13
CT 2-ways	-	N/A	5	6
CT 3-ways	-	-	N/A	8

5 DISCUSSION

Our results regarding the requirement coverage show that all the techniques, except 2-ways, achieved 100% requirement coverage. The techniques differed more with respect to the fulfilment of different conditions for MC/DC. Overall, the MC/DC condition that evaluates the effectiveness of the test suites in terms of 'independent effect on the outcome of a decision' was the least dominant condition fulfilled by all the test suites. The rest of the MC/DC conditions were met 100% by MBT, manual testing and higher strength CT (3- and 4-ways).

Our results have also shown that MBT gave far greater number of unique test cases when compared with CT and manual testing. In terms of efficiency, MBT is found to be better than manual testing but worse than CT. For CT, we observed that increasing the strength

also increased the number of unique test cases along with increasing the overall number of test cases. CT was the most efficient of all the techniques but the absence of knowledge regarding expected outcomes makes CT dependent on tester's skills.

In an industrial setting where manual testing is performed, our results indicate the MBT and CT can both add important test cases, with MBT providing the most number of additions. Whether such additions are fault revealing or not is left as a future investigation but at least in terms of MC/DC coverage, MBT test suite also has given evidence in support of its advantage. Our results also showed that 2-way CT strategy does not perform better than higher strength CT strategies and 90% of its test cases are covered by MBT as well as higher strength CT gives more unique test cases, thus if CT is to be adopted, our results support utilising greater than 2-way strength CT.

What is also clear from our study is that there is no silver bullet when it comes to testing of safety-critical systems in terms of effectiveness and efficiency. A mix of techniques with an understanding of advantages and disadvantages in terms of efficiency and effectivness seems like the best advice for industrial practitioners. Additionally, as with any new techniques, there are challenges at the start, both with MBT [11] and CT [22], [24]. From a research perspective, hybrid testing techniques that can combine the best of CT and MBT [19] seems like an interesting avenue to take.

6 VALIDITY THREATS

This section deals with the threats which can affect the validity of this study and the measures we took to address them.

6.1 Internal Validity

One of the main threats to internal validity is the validation and conformance of the model with system requirements. However, we have alleviated this factor by creating the model in multiple iterations and continuously consulting with the testing team at Alstom Transport AB, which eventually confirmed it as correct. Another issue relates to the indirect cost of testing techniques, such as maintenance cost of test suites and development time of tools like TIGER. In this study, we have not considered the effect of indirect costs and therefore, considering both direct and indirect costs may affect efficiency results. We have spent a fair amount of time to analyse all the test suites for coverage and efficiency manually; a manual analysis of such large amount of data can result in small errors but these should not be large to affect our results in any meaningful way.

6.2 Reliability and External Validity

The threats related to reliability and external validity include generalization of the MBT model, size of state space, complexity of the system, impact of human experience, generator algorithms and modelling notations. We are working with modelling and generation of test suites at system level for the subsystems of TCMS developed at Alstom Transport AB, so it contains particularities for the generation of test suites that may not be applicable to other domains. However, for the replication of this study in a similar domain, we have tried to provide enough information about the experimentation setup. We cautiously argue that if another researcher with similar experience of the testing domain and modelling will replicate this study, similar results should be produced, however different modelling languages, testing tools and generator algorithms may affect the results. Moreover, for this study, we have used a part of the TCMS system with their actual number of parameters and constraints. Nevertheless, more case studies are required to generalize the results of this study to larger systems.

6.3 Construct Validity

The operational measures used in this study for cost estimation of testing techniques were inspired by taxonomy of MBT provided by Kramer et al. [23] and industrial practices at Alstom Transport AB, Sweden. Moreover, the measure for requirement and MC/DC coverage for comparative analysis of test suites were determined by a thorough investigation of literature and industrial applicability of testing strategies.

7 CONCLUSION AND FUTURE WORK

This paper provides a comparative analysis between MBT, CT and manual testing techniques in terms of MC/DC and requirement coverage. It also evaluates the efficiency of testing techniques and determines the differences and overlaps of test suites generated by each testing technique. The experimentation results based on an industrial case study showed that regardless of test objectives used to develop test cases for each technique (i.e. boundary value analysis (BVA) and equivalence partitioning (EP) for manual testing, edge coverage in MBT and t-ways interactions of parameters in CT), each test suite achieved a substantial level of MC/DC coverage. However, the test suite generated using MBT provided a higher MC/DC coverage. MBT-generated test suite contained approximately 71% of similar test cases, on average, generated by other testing techniques as well as highest number of unique test cases, which also had an observable effect on MC/DC adequacy. Furthermore, the analysis also showed that MBT-generated test suite is highly relevant to manual specification-based testing in terms of complete test case generation due to model's conformance with requirements. Hence, we argue that MBT-generated test suite is most likely to uncover system-level faults and it could be used to improve manual testing. On the other hand, CT was the most efficient technique when compared to MBT and manual testing but exponential growth of test cases while achieving higher MC/DC could affect its efficiency. Also, MBT-generated test suite contains redundant test cases and their exclusioncan reduce the execution time, consequently efficiency could be improved.

In future, we intend to perform a thorough evaluation of the generated test suites in terms of fault detection effectiveness and to investigate approaches to reduce test suite generated by MBT.

ACKNOWLEDGMENTS

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 XIVT project and from the ECSEL Joint Undertaking (JU) under grant agreement No 101007350.

REFERENCES

[1] Wasif Afzal, Ahmad Nauman Ghazi, Juha Itkonen, Richard Torkar, Anneliese Andrews, and Khurram Bhatti. 2015. An experiment on the effectiveness and efficiency of exploratory testing. *Empirical Software Engineering* 20 (2015), 844–878.

[2] 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. 2013. An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software* 86, 8 (2013), 1978–2001.

[3] Andrea Bombarda and Angelo Gargantini. 2020. An Automata-Based Generation Method for Combinatorial Sequence Testing of Finite State Machines. In *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 157–166.

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

[5] Jörg Brauer, Jan Peleska, and Uwe Schulze. 2012. Efficient and trustworthy tool qualification for model-based testing tools. In *IFIP International Conference on Testing Software and Systems*. Springer, 8–23.

[6] Cristian Cadar, Daniel Dunbar, Dawson R Engler, et al. 2008. Klee: unassisted and automatic generation of high-coverage tests for complex systems programs.. In *OSDI*, Vol. 8. 209–224.

[7] Yin Chen, Sven Linder, and Jonas Wigstein. 2019. An Approach of Creating Component Design Specification for Safety-Related Software in Railway. In *2019 Annual Reliability and Maintainability Symposium (RAMS)*. IEEE, 1–4.

[8] John Joseph Chilenski and Steven P Miller. 1994. Applicability of modified condition/decision coverage to software testing. *Software Engineering Journal* 9, 5 (1994), 193–200.

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

[10] Siddhartha R Dalal, Ashish Jain, Nachimuthu Karunanithi, JM Leaton, Christopher M Lott, Gardner C Patton, and Bruce M Horowitz. 1999. Model-based testing in practice. In *Proceedings of the 21st international conference on Software engineering*. 285–294.

[11] Arilo C. Dias-Neto and Guilherme H. Travassos. 2010. A Picture from the Model-Based Testing Area: Concepts, Techniques, and Challenges. In *Advances in Computers*, Marvin V. Zelkowitz (Ed.). Advances in Computers, Vol. 80. Elsevier, 45–120.

[12] Arnaud Dupuy and Nancy Leveson. 2000. 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)*, Vol. 1. IEEE, 1B6–1.

[13] Mounia El qortobi, Amine Rahj, Jamal Bentahar, and Rachida Dssouli. 2020. Test Generation Tool for Modified Condition/Decision Coverage: Model Based Testing. In *Proceedings of the 13th International Conference on Intelligent Systems: Theories and Applications*. 1–6.

[14] Eduard Enoiu, Daniel Sundmark, Adnan Čaušević, and Paul Pettersson. 2017. A Comparative Study of Manual and Automated Testing for Industrial Control Software. In *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. 412–417. https://doi.org/10.1109/ICST.2017.44

[15] Miraldi Fifo, Eduard Enoiu, and Wasif Afzal. 2019. On measuring combinatorial coverage of manually created test cases for industrial software. In *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 264–267.

[16] Svetoslav R Ganov, Chip Killmar, Sarfraz Khurshid, and Dewayne E Perry. 2008. Test generation for graphical user interfaces based on symbolic execution. In *Proceedings of the 3rd international workshop on Automation of software test*. 33–40.

[17] Gregory Gay, Ajitha Rajan, Matt Staats, Michael Whalen, and Mats PE Heimdahl. 2016. The effect of program and model structure on the effectiveness of mc/dc test adequacy coverage. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 25, 3 (2016), 1–34.

[18] Jon Hagar, Rick Kuhn, Raghu Kacker, and Tom Wissink. 2014. Introducing combinatorial testing in a large organization: Pilot project experience report. In *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops*. IEEE, 153–153.

[19] Jon Hagar, Rick Kuhn, Raghu Kacker, and Tom Wissink. 2014. Introducing Combinatorial Testing in a Large Organization: Pilot Project Experience Report. In *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops*.

[20] Kelly J Hayhurst. 2001. *A practical tutorial on modified condition/decision coverage*. DIANE Publishing.

[21] Hadi Hemmati, Syed S Arefin, and Howard W Loewen. 2018. 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)*. IEEE, 256–265.

[22] Linghuan Hu, W. Eric Wong, D. Richard Kuhn, and Raghu N. Kacker. 2020. How does combinatorial testing perform in the real world: an empirical study. *Empirical Software Engineering* 25 (2020), 2661–2693.

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

[24] Peter M. Kruse, Nelly Condori-Fernández, Tanja E.J. Vos, Alessandra Bagnato, and Etienne Brosse. 2013. Combinatorial Testing Tool Learnability in an Industrial Environment. In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*.

[25] D Richard Kuhn, Raghu N Kacker, Yu Lei, et al. 2015. Combinatorial coverage as an aspect of test quality. *CrossTalk* 28, 2 (2015), 19–23.

[26] Rick Kuhn and Raghu Kacker. 2011. Practical combinatorial (t-way) methods for detecting complex faults in regression testing. In *2011 27th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 599–599.

[27] Kiran Lakhotia, Phil McMinn, and Mark Harman. 2009. Automated Test Data Generation for Coverage: Haven't We Solved This Problem Yet?. In *2009 Testing: Academic and Industrial Conference - Practice and Research Techniques*. 95–104. https://doi.org/10.1109/TAICPART.2009.15

[28] Hareton KN Leung and Lee J White. 1991. A cost model to compare regression test strategies.. In *ICSM*, Vol. 91. 201–208.

[29] Dong Li, Linghuan Hu, Ruizhi Gao, W Eric Wong, D Richard Kuhn, and Raghu N Kacker. 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)*. IEEE, 297–303.

[30] W. Li, F. Le Gall, and N. Spaseski. 2017. A survey on model-based testing tools for test case generation. In *International Conference on Tools and Methods for Program Analysis*. Springer.

[31] Arthur Marques, Franklin Ramalho, and Wilkerson L Andrade. 2014. Comparing model-based testing with traditional testing strategies: An empirical study. In *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops*. IEEE, 264–273.

[32] Ahmed Mateen, Qingsheng Zhu, and Salman Afsar. 2018. Comparitive Analysis of Manual vs Automotive Testing for Software Quality. In *Proceedings of the 7th International Conference on Software Engineering and New Technologies*. 1–7.

[33] Stefan Mohacsi, Michael Felderer, and Armin Beer. 2015. Estimating the Cost and Benefit of Model-Based Testing: A Decision Support Procedure for the Application of Model-Based Testing in Industry. In *2015 41st Euromicro Conference on Software Engineering and Advanced Applications*. 382–389. https://doi.org/10.1109/SEAA.2015.18

[34] Cu D Nguyen, Alessandro Marchetto, and Paolo Tonella. 2012. Combining model-based and combinatorial testing for effective test case generation. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis*. 100–110.

[35] Sebastian Oster, Ivan Zorcic, Florian Markert, and Malte Lochau. 2011. MoSo-PoLiTe: tool support for pairwise and model-based software product line testing. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*. 79–82.

[36] Mike Papadakis and Nicos Malevris. 2010. Automatic mutation test case generation via dynamic symbolic execution. In *2010 IEEE 21st International Symposium on Software Reliability Engineering*. IEEE, 121–130.

[37] Rudolf Ramler, Theodorich Kopetzky, and Wolfgang Platz. 2012. Combinatorial Test Design in the TOSCA Testsuite: Lessons Learned and Practical Implications. In *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*. 569–572. https://doi.org/10.1109/ICST.2012.142

[38] Sanjai Rayadurgam and Mats Heimdahl. 2003. Generating MC/DC adequate test sequences through model checking. (2003).

[39] Mary Sánchez-Gordón, Laxmi Rijal, and Ricardo Colomo-Palacios. 2020. Beyond Technical Skills in Software Testing: Automated versus Manual Testing. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*. 161–164.

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

[41] Ossi Taipale, Jussi Kasurinen, Katja Karhu, and Kari Smolander. 2011. Trade-off between automated and manual software testing. *International Journal of System Assurance Engineering and Management* 2, 2 (2011), 114–125.

[42] M. Utting and B. Legeard. 2010. *Practical model-based testing: a tools approach*. Elsevier.

[43] Erik Van Veenendaal, Dorothy Graham, and Rex Black. 2008. "Foundations of Software Testing: ISTQB Certification. *Cengage Learning EMEA* (2008), 30.

[44] Sergiy Vilkomir, Aparna Alluri, D Richard Kuhn, and Raghu N Kacker. 2017. Combinatorial and MC/DC coverage levels of random testing. In *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 61–68.

[45] Michael Wagner, Kristoffer Kleine, Dimitris E. Simos, Rick Kuhn, and Raghu Kacker. 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). 191–200. https://doi.org/10.1109/ICSTW50294.2020.00041

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

[47] Muhammad Nouman Zafar, Wasif Afzal, and Eduard Enoiu. 2021. Towards a Workflow for Model-Based Testing of Embedded Systems. In *Proceedings of the 12th International Workshop on Automating TEST Case Design, Selection, and Evaluation*. Association for Computing Machinery, New York, NY, USA.

[48] Muhammad Nouman Zafar, Wasif Afzal, Eduard Paul Enoiu, Athanasios Stratis, Aitor Arrieta, and Goiuria Sagardui. 2021. Model-Based Testing in Practice: An Industrial Case Study using GraphWalker. In *Innovations in Software Engineering Conference 2021*. http://www.es.mdh.se/publications/6101-

[49] Muhammad Nouman Zafar, Wasif Afzal, Eduard Paul Enoiu, Athanasios Stratis, and Ola Sellin. 2021. A Model-Based Test Script Generation Framework for Embedded Software. In *The 17th Workshop on Advances in Model Based Testing*. http://www.es.mdh.se/publications/6172-