Software Qual J (2018) 26:1327–1372
DOI 10.1007/s11219-017-9386-2

Model-based testing for software safety: a systematic mapping study

Havva Gulay Gurbuz[1] · Bedir Tekinerdogan[1]

Published online: 25 September 2017

Abstract Testing safety-critical systems is crucial since a failure or malfunction may result in death or serious injuries to people, equipment, or environment. An important challenge in testing is the derivation of test cases that can identify the potential faults. Model-based testing adopts models of a system under test and/or its environment to derive test artifacts. This paper aims to provide a systematic mapping study to identify, analyze, and describe the state-of-the-art advances in model-based testing for software safety. The systematic mapping study is conducted as a multi-phase study selection process using the published literature in major software engineering journals and conference proceedings. We reviewed 751 papers and 36 of them have been selected as primary studies to answer our research questions. Based on the analysis of the data extraction process, we discuss the primary trends and approaches and present the identified obstacles. This study shows that model-based testing can provide important benefits for software safety testing. Several solution directions have been identified, but further research is critical for reliable model-based testing approach for safety.

Keywords Model-based testing · Model-driven testing · Software safety · Systematic mapping study

1 Introduction

Currently, an increasing number of systems are controlled by software and rely on the correct operation of the software. In this context, a safety-critical system is

✉ Havva Gulay Gurbuz
havva.gurbuz@wur.nl

Bedir Tekinerdogan
bedir.tekinerdogan@wur.nl

[1] Information Technology Group, Wageningen University, Wageningen, The Netherlands

defined as a system in which the malfunctioning of software could result in death, injury, or damage to environment. Software can be considered safe if it does not produce an output that causes a catastrophic event for the system. Several methods, processes, and models are developed in order to make the software safe. System safety engineering is the application of engineering and management principles, criteria, and techniques to optimize all aspects of safety within the constraints of operational effectiveness, time, and cost throughout all phases of the system life cycle.

Testing the software of safety-critical systems is crucial since a failure or malfunction may result in death or serious injury to people, or loss or severe damage to equipment or environmental harm. Software testing of safety-critical systems can be stated as the process of validating and verifying that a system meets the safety requirements that guided its design and development and likewise satisfies the needs of stakeholders. Testing usually includes the process of executing a program or application with the intent of finding software defects. Software defects may result in an error that could in the end cause a failure that could be safety-critical. An important challenge in testing is the derivation of test cases that can identify the potential faults. In large-scale and complex software systems, testing can be laborious and time consuming when it is done manually.

Model-based testing (MBT) adopts models of a system under test and/or its environment for designing and optionally also executing artifacts to perform software testing or system testing. Using explicit models helps to structure the process of deriving tests and support the reuse, reproduction, and documentation of test cases. In addition, MBT enables the automated production and execution of test cases, which on its turn reduces the cost and time of testing and increases the quality of test cases (Rafi et al. 2012). MBT has been applied for testing both functional and nonfunctional properties. In this paper, we focus on the application of MBT for testing safety properties. Several approaches have been provided for this in the literature, but no effort has been provided yet to provide an overall analysis of the studies in the literature. The overall objective of this paper is thus to provide a systematic mapping study (SMS) to systematically identify, analyze, and describe the state-of-the-art advances in model-based testing for software safety.

The SMS is conducted as a multi-phase study selection process using the published literature in major software engineering journals and conference proceedings. We have reviewed 751 papers that were discovered using a well-planned review protocol, and 36 of them were assessed as primary studies related to our research questions. Based on the analysis of data extraction process, we discuss the primary trends and approaches and present the identified obstacles.

For researchers, this SMS gives an overview of the reported model-based testing for software safety with the strength of empirical evidence of the identified approaches. Practitioners may benefit from the SMS by identifying the strengths and weaknesses of the approaches as well as the remaining important challenges.

The remainder of the paper is organized as follows. Section 2 provides the preliminaries including background of model-based testing and systematic mapping study. Section 3 gives the details of SMS method adopted in this study. Section 4 provides the result of the SMS study. Section 5 presents the related work and finally Section 6 concludes the paper.

2 Background

2.1 Model-based testing

The IEEE Software Engineering Body of Knowledge (SWEBOK 2004) defines testing as an activity performed for evaluating product quality, and for improving it, by identifying defects and problems (Bourque and Dupuis 2004). In contrast to static analysis techniques, testing requires the execution of the program with specific input values to find failures in its behavior. In general, exhaustive testing is not possible or practical for most real programs due to the large number of possible inputs and sequences of operations. Because of the large set of possible tests, only a selected set of tests can be executed within feasible time limits. As such, the key challenge of testing is how to select the tests that are most likely to expose failures in the system. Moreover, after the execution of each test, it must be decided whether the observed behavior of the system was a failure or not. This is called the oracle problem.

In the traditional test process, the design of test cases and the oracles as well as the execution of the tests are performed manually. This manual process is time consuming and less tractable for the human tester. MBT relies on models of system requirements and behavior to automate the generation of the test cases and their execution. The general process for MBT is shown in Fig. 1 (Utting et al. 2006). Based on the *test requirements* and the *test plan*, a *test model* is constructed. A model is usually an abstract, partial presentation of the desired behavior of a *system under test* (SUT). The test model is used to generate test cases that together form the *abstract test suite*. Because of that, there are usually an infinite number of possible tests; usually test selection criteria are adopted to select the proper test cases. For example, different model coverage criteria, such as all-transitions, can be used to derive the corresponding test cases. The resulting test cases lack the detail needed by the SUT and as such are not directly executable. In the third step, the abstract test suite is transformed to a concrete or executable test suite. This is typically done using a transformation tool, which translates each abstract test case to an executable test case. An advantage of the separation between abstract test suite and concrete test suite is the platform and language independence of the abstract test cases. The same abstract test case can be reused in different test execution environments. In the fourth step, the concrete test cases are executed on the SUT. A distinction is made between online MBT and offline MBT. In online MBT, the concrete test cases are executed as they are produced. In offline MBT, the test cases are produced before the execution. The test execution will result in a report that contains the outcome of the execution of the test cases. In the final, fifth step, these results are analyzed and if needed corrective actions are taken. Hereby, for each test that reports a failure, the cause of the failure is determined and the program (or model) is corrected.

2.2 Systematic mapping studies

A systematic mapping study (also referred to as scoping study) is a practice that based on the evidence-based research that was mostly used in the field of medicine and has been adopted by software engineering. The SMS is a methodology to give an overview of a research area through classification and counting contributions in relation to the categories of that classification (Petersen et al. 2015). The results of the SMS are often illustrated by visual maps to provide an overview of the state-of-art on a specified research area.

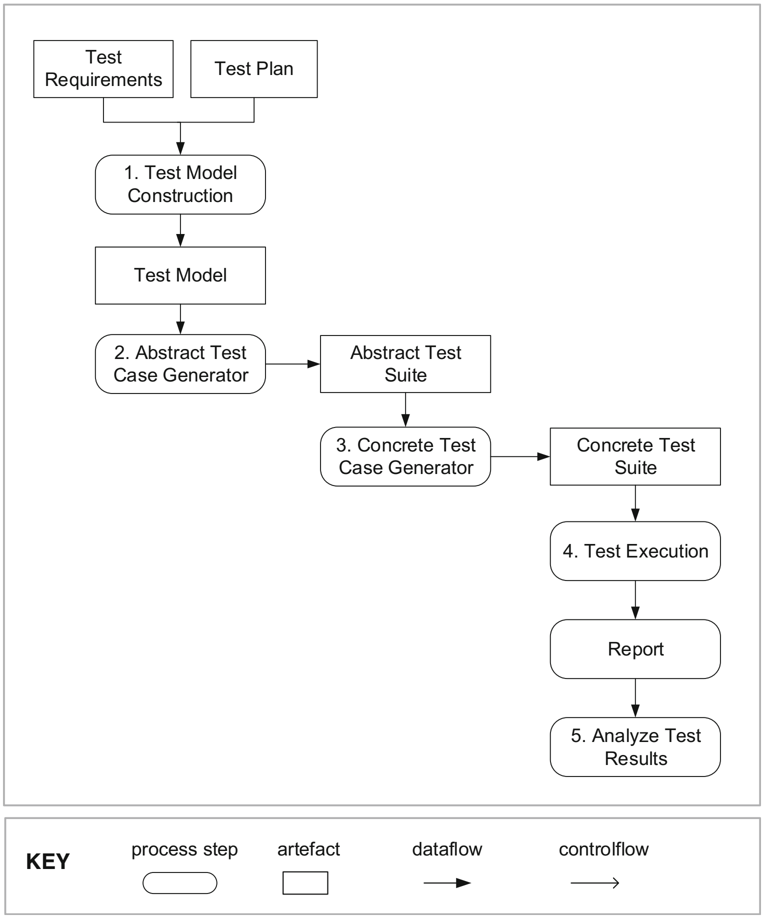

Fig. 1 General process for MBT

A systematic mapping study is conducted to investigate a relatively broad topic and aims to identify, analyze, and structure the goals, methods, and contents of previous primary studies. In our study, we aim to classify and analyze the literature and provide an overview of existing research directions regarding the model-based testing for software safety. Therefore, a SMS is a suitable research method for our research.

3 Research method

We carry out the SMS to provide an overview of existing research directions regarding the MBT for software safety by following the guidelines and process proposed by (Petersen et al. 2015). The remainder of this section describes the steps of this process.

3.1 Mapping study protocol

Before conducting the systematic mapping study, firstly we developed a mapping study protocol which defines the methods that will be used to perform a specific systematic mapping study. The

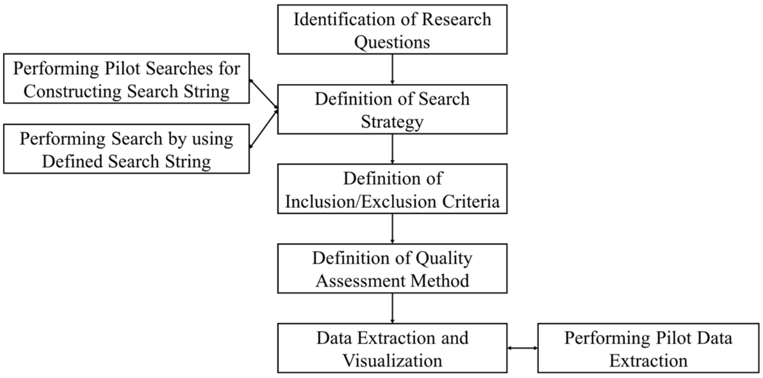
Identification of Research Questions
Performing Pilot Searches for Constructing Search String
Definition of Search Strategy
Performing Search by using Defined Search String
Definition of Inclusion/Exclusion Criteria
Definition of Quality Assessment Method
Data Extraction and Visualization
Performing Pilot Data Extraction

Fig. 2 Mapping study protocol

pre-defined protocol reduces the researcher bias. The adopted mapping study protocol is shown in Fig. 2. Firstly, we specified our research questions (discussed in Section 3.2) based on the objectives of this SMS. Then, we defined the search strategy and search scope to specify the time span and the venues that we considered to conduct our study (explained in Section 3.3). In the search strategy, we devised the search strings that were formed after performing deductive pilot searches. A well-defined search string brings the appropriate search results that will come to a successful conclusion in terms of sensitivity and precision rates. After we defined the search strategy, we specified the study selection criteria (Section 3.4) which were used to determine which studies are included in, or excluded from, the systematic mapping study. We piloted the selection criteria on a number of primary studies. We screened the primary studies at all phases on the basis of inclusion and exclusion criteria. In addition, we performed peer reviews throughout the study selection process. After this step, we conducted quality assessment in which the primary studies that resulted from the search process were screened based on quality assessment checklists and procedures (Section 3.5). Once the final set of preliminary studies was defined, we specified the data extraction strategy which defines how the information required from each study is obtained (Section 3.6). For this, we developed a data extraction form that was defined after a pilot study. In the final step, we present the data extraction results.

3.2 Research questions

The most important part of any systematic mapping study is specifying the research questions clearly and explicitly. Research questions drive the subsequent parts of the systematic mapping study. Hence, asking the right question is crucial to derive the relevant findings properly. The more precise research questions bring the more accurate findings. In this context, research questions need to be meaningful and important to both practitioners and researchers. In this paper, we are interested in investigating empirical studies that are done about model-based testing for software safety. In order to examine the evidence of model-based testing for software safety, we define the following research questions:

RQ.1: In which application domains is model-based testing applied?

With this research question, we aim to identify the different application domains in which model-based testing has been applied. This will highlight the current scope and applicability of model-based testing.

RQ.2: What are the existing research directions within model-based testing for software safety?
RQ.2.1: What is the motivation for adopting model-based testing for software safety?
RQ.2.2: What are the proposed solutions in model-based testing for software safety?
RQ.2.3: What are the research challenges in model-based testing for software safety?

With this research question, we aim to make the current research directions explicit for model-based testing for software safety in particular. Related to this, we will identify the motivation for adopting model-based testing for software safety, the current solutions, and the research challenges.
RQ.3: What is the strength of evidence of the study?
Elaborating on RQ2, we will further explore whether the proposed solution approaches have been empirically justified. This is important to highlight scientific evidence and as such identify the proven solutions.

3.3 Search strategy

The aim of the SMS is finding as many primary studies relating to the research questions as possible using a well-planned search strategy. In this section, we describe our search strategy by explaining search scope, adopted search method, and search string.

3.3.1 Scope

Our search scope consists of two dimensions that are publication period and publication venues. In terms of publication period (time), we include the papers that were published over the period of 1992 and August 2015. We use the following well-known search databases to search studies: Scopus, IEEE Xplore, ACM Digital Library, Wiley Inter Science Journal Finder, Science Direct, Springer Link, and ISI Web of Knowledge. Our targeted search items are journal papers, conference papers, articles, and workshop papers.

3.3.2 Search method

To perform the search, in the selected databases, we use both manual and automatic search. Automatic search is realized through entering search strings on the search engines of the electronic data source. Manual search is realized through manually browsing the conferences, journals, or other important sources.

The outcome of a search process can easily lead to a very high number of papers. In this respect, for the search process it has been pointed out that the relevant studies are selected (high recall) while the irrelevant ones are ruled out (high precision). Usually depending on the objectives of an SMS, one of the criteria (recall or precision) can be favored and used by the investigators. Hereby, a search strategy that focuses on high recall only can require too much manual effort of dealing with irrelevant articles whereas a precise search strategy can unavoidably miss many relevant articles. Zhang et al. (Zhang et al. 2011) proposes the solution called as quasi-gold standard (QGS) in order to identify the relevant studies while reducing the number of irrelevant ones as much as possible. Hereby, before defining the search query first, a manual survey of publications is carried out in which the employed search strings are analyzed and elicited. The resulting search strings are then fed into the search query aiming to find the optimal set with respect to the recall and precision rates.

We adopted this approach to reveal better keywords in designating search strings, and likewise to achieve high recall rate and high precision rate. The primary studies, which we manually selected in reliance upon our knowledge of topic, were analyzed in order to elicit better keywords that would optimize the retrieval of relevant material. The analysis of the articles in the QGS was carried out using word frequency and statistical analysis tools. First, the term frequency, inverse document frequency (TF*IDF) algorithm (Leskovec et al. 2014), was operated on the titles and abstracts of the QGS papers. As stated by (Zhang et al. 2011), full text analysis would mislead us into thinking inaccurate keywords as true indicators because of the titles in the reference section. In addition, the keywords of authors were manually examined to enhance the representative set of words observed. Finally, a definite set of search strings was obtained.

3.3.3 Search string

For the automated search, we construct a search string after performing a number of pilot searches to get relevant studies at utmost level. Since each electronic data source requires its own expression format to enter the search queries, we have defined these separately. However, although the syntax of the queries is necessarily different, we have ensured that the queries are semantically equivalent. To create more complex queries, we use the OR and AND operators. The following text represents the search string is defined for IEEE Xplore database:

(("Abstract": model based software test OR "Abstract": model driven software test) AND "Abstract": safety)

The search strings for other electronic databases are given in Appendix Table 11 Search Strings. The result of the overall search process after applying the search queries is given in the second column of Table 1. As shown in the table, we identify in total 751 papers at this stage of the search process. The third column of the table presents the number of papers where the full texts of papers are available. Since some studies can be shown in different electronic databases multiple times, we apply a manual search to find duplicate publications. After applying the last stage of the search process 36 papers are left.

3.4 Study selection criteria

Since the search query strings have a broad scope to ensure that any important documents are not omitted, the automated search can easily lead to a large number of documents. In accordance with the SMS guidelines, we further apply two exclusion criteria on the large-sized sample of papers in the first stage. The overall exclusion criteria that we use are as follows:

- EC1: Papers which do not have full text
- EC2: Duplicate publications found in different search sources

- EC3: Papers are not written in English
- EC4: Papers do not relate to software safety
- EC5: Papers do not relate to model-based/model-driven testing
- EC6: Papers do not explicitly discuss safety
- EC7: Experience and survey papers
- EC8: Papers do not validate the proposed study

The exclusion criteria are applied manually. After applying these criteria, 36 papers of the 751 papers are selected.

3.5 Study quality assessment

In addition to general inclusion/exclusion criteria, we also consider to assess the quality of primary studies. The main goals of the quality assessment step are providing more detailed inclusion/exclusion criteria, determining the importance of individual studies once results are being synthesized, guiding the interpretation of findings and leading recommendations for further research. We consider the quality assessment as part of our data extraction process and use the result of the assessment while providing an answer to RQ3.

We develop a quality assessment based on quality instruments which are checklist of factors that need to be assessed for each study (Kitchenham and Charters 2007). The quality checklist is derived by considering the factors that could bias study results. While developing our quality assessment, we adopt the summary quality checklist for quantitative studies and qualitative studies which is proposed on Kitchenham and Charters (2007). Table 2 presents the quality checklist. Since the aim is ranking studies according to an overall quality score, we deploy the items in the quality checklist on a numeric scale. We use the three-point scale and assign scores (yes = 1, somewhat = 0.5, no = 0) to each criterion. The results of assessment are given in Appendix Table 13 Study Quality Assessment Form. These results are used in order to support data extraction step.

Table 1 Overview of search results and study selection

Source	Number of included studies after applying search query	Number of included studies after EC1-EC3 is applied	Number of included studies after EC4-EC8 is applied
IEEE Xplore	53	40	9
ACM Digital Library	10	5	0
Wiley Interscience	36	18	0
Science Direct	7	7	5
Springer	466	325	18
ISI Web of Knowledge	32	7	0
Scopus	147	43	4
Total	751	445	36

Table 2 Quality checklist

Number	Assessment category	Question
Q1	Quality of reporting	Are the aims of the study is clearly stated?
Q2		Are the scope and context of the study clearly defined?
Q3		Is the proposed solution clearly explained and validated by an empirical study?
Q4	Rigor	Are the variables used in the study likely to be valid and reliable?
Q5		Is the research process documented adequately?
Q6		Are the all study questions answered?
Q7	Credibility	Are the negative findings presented?
Q8		Are the main findings stated clearly in terms of creditability, validity and reliability?
Q9	Relevance	Do the conclusions relate to the aim of the purpose of study?
Q10		Does the report have implications in practice and results for model-based testing for software safety?

3.6 Data extraction and visualization

In order to extract data needed to answer the research questions, we read the full texts of 36 selected primary studies. We designed a data extraction form to collect all the information needed to address the review questions and the study quality criteria. The data extraction form includes standard information such as study ID, date of extraction, year, authors, repository, publication type, and space for additional notes. In order to collect information directly related to answering research questions, we have added some fields, such as targeted domain, motivation for study, solution approach, constraints/limitations of approach, and findings. All related fields to research questions are shown in Table 3. We have a record of the extracted information in a spreadsheet to support the process of synthesizing the extracted data. In Appendix Table 12, we have provided the details of the data extraction form. We present the data extraction results in Section 4 by explaining them textually and visually. In order to make data extraction results easy to understand, we use tables and charts while presenting the distribution of the primary studies according to extracted data.

Table 3 Data extraction

Research question		Data extracted
RQ1		Targeted domain
RQ2	RQ2.1	Motivation for study
		Main theme of study
	RQ2.2	Requirement specification language
		Safety model specification language
		Method for generating models from requirements
		Type of generated test elements (test case, test oracle, test data etc.)
		Solution approach for test element
		Abstract test case generation
		Test selection criteria
		Test case specification language
		Method for test execution
		Contribution type
	RQ2.3	Constraints/limitation of proposed solution
		Findings
RQ3		Assessment approach

4 Results

4.1 Research methods

It is very important to conduct empirical studies with well-defined research methodologies to ensure the reliability and validity of the findings. Primary studies are expected to explicitly define and report the used research methodology. In Table 4, we provide the information about the type of research methods used in the 36 selected primary studies. We consider the categorization in Wohlin et al. (2012) to extract research method during the review process. Wohlin et al. (2012) define three main research methods, surveys, case studies, and experiments, for empirical studies. Since we do not include surveys in our work, we only consider case study and experiment as research methods to categorize primary studies. It can be observed that *case study* is the dominant method used to evaluate the model-based testing for software safety approaches. Case studies are conducted to understand, to explain, or to demonstrate the capabilities of a new technique, method, tool, process, technology, or organizational structure (Wohlin et al. 2012). In addition, Table 4 shows that in the reviewed primary studies, experiments are also used to analyze and assess the proposed approaches. Experiments are conducted to investigate of a testable hypothesis in which conditions are set up to isolate the variables of interest and test how they affect certain measurable outcomes.

Besides the research method, we also extract the evidence type to indicate that primary studies validate their solution approaches using academic or industrial evidence. Fifteen (42%) of the primary studies use industrial evidence to validate their proposed model-based testing methods for software safety. Twenty-one (52%) of the primary studies illustrate their solution approach using academic evidence. In Table 5, we provide the information about the studies and their evidence type.

4.2 Methodological quality

In this section, we present the quality of selected primary studies. For this purpose, we try to address methodological quality in terms of relevance, quality of reporting, rigor, and assessment of credibility by using the quality checklist that is defined in Table 2. Therefore, we

Table 4 Distribution of studies over research method

Research method	Studies	Number	Percent
Case study	[1], [2], [4], [5], [6], [7], [8], [9], [10], [11], [12], [14], [16], [17], [18], [21], [22], [24], [26], [28], [29], [31], [33], [34], [35], [36]	26	72
Experiment	[3], [13], [15], [19], [20], [23], [25], [27], [30], [32]	10	28

Table 5 Evidence type for selected studies

Evidence type	Studies	Number	Percent
Academic	[3], [5], [6], [7], [8], [9], [10], [12], [13], [14], [15], [17], [18], [23], [24], [26], [29], [30], [32], [34], [36]	21	58
Industrial	[1], [2], [4], [11], [16], [19], [20], [21], [22], [25], [27], [28], [31], [33], [35]	15	42

group the first three questions of the checklist for the quality of reporting, the ninth and tenth questions for the relevance, the fourth, fifth, and sixth questions for rigor, and the seventh and eighth questions for assessment of credibility of evidence. In Appendix Table 13 Study Quality Assessment Form, we present the result of quality checklist.

For the quality of reporting based on the result of first three questions, two studies (6%) are weak with score 2, 22 (61%) of the studies are average with score 2.5, and 12 (33%) of the primary studies are good with score 3.

In order to assess the primary studies' quality according to the trustiness of findings, we assess the rigor of studies. The quality score of rigor of studies is calculated based on the result of fourth, fifth, and sixth questions. Five (14%) of the primary studies have poor quality with score 1.5. Five (14%) of the primary studies have average quality with score 2. Eighteen (50%) primary studies are good according to rigor quality with score 2.5. Further, eight papers (22%) of the primary studies are assessed as top quality in terms of rigor with score 3.

As another methodological quality measure, we assess the relevance of the selected primary studies. The relevance quality scores are calculated based on the evaluation of the ninth and tenth questions. Thirty-nine percent of the primary studies with score 2 are directly relevant to the model-based software safety testing, and 61% of the primary studies with scores 1 and 1.5 are to some extent relevant to the field.

In order to assess the primary studies in terms of credibility, validity, and reliability of positive and negative findings and major conclusions of the primary studies, the quality score is calculated based on results of seventh and eighth questions. According to our evaluation, there is no primary study that has full credibility of evidence. Considering the score 1.5 as first rate, 4 (11%) of the primary studies are good with score 1.5. The studies having score 1 were treated as fair, and 18 (50%) of the primary studies fall into this category. Fourteen studies (39%) have poor quality score according to their credibility of evidence.

Finally, we summarize this section by giving the overall methodological quality scores. In Fig. 3, the total quality of scores is presented in terms of our four criteria: quality of reporting, relevance, rigor, and credibility of evidence. Considering the scores 9 and 9.5 as high, 5 (14%) of the primary studies have high quality. Eighteen (50%) primary studies having scores (7.5, 8.5) have good quality. Thirteen (36%) of the studies having scores (5.5, 7) have poor quality.

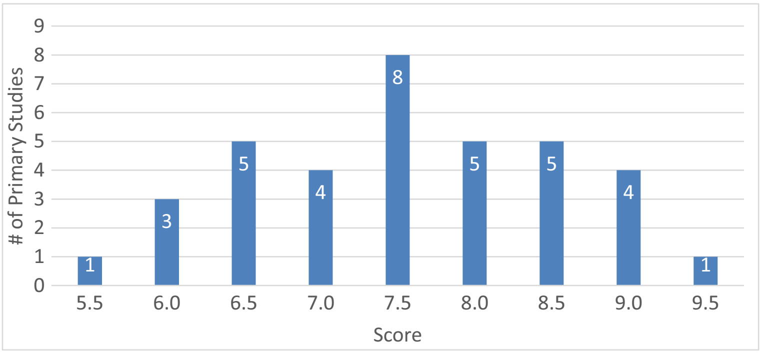

Fig. 3 Quality assessment scores

4.3 RQ.1–MBT in application domains

RQ.1: In which application domains is model-based testing applied?

In order to answer this research question, we analyze the targeted domains of the 36 selected primary studies separately. In Table 6, we present the categories of targeted domain that we extracted. There are seven main application domains namely, automotive, railway, nuclear, robotics, automation, medical, and power consumption (Fig. 4).

As shown in Table 6, the category automotive includes ten subcategories. [12], [17], [23] and [30] apply the model-based testing on alarm systems for cars. [12] perform model-based testing on software product family of automotive domain. [13] and [30] discuss the cruise control system that automatically controls the speed of a car. In [9], a model-based approach for test case generation is described for embedded control systems for cars. [1] apply the model-based testing on the embedded system application for cars. [3] discuss the control system in vehicles. [27] apply model based testing on landing gear system. In [31], they illustrate their approach on the turn indicator of automotive system. [30] illustrate the model-based testing on window controller. In [32], they illustrate their approach on anti-slip regulation/anti-lock braking system (ASR/ABS). In [26], they demonstrate the proposed solution on an automotive operating system. As shown in the table, [36] illustrate the proposed solution on several subcategories. They use cruise control, landing gear system, and different controller systems.

In the domain *Railway*, model-based testing is applied on seven different subcategories listed in Table 6. [8], [16] and [29] discuss the railway interlocking system that prevents trains from colliding and drilling while allowing trains movements. [2], [4], [7] and [21] discuss the train control system represents an important part of the railway operations management system. [5] applies the model-based testing on railway onboard system which is responsible for implementation of over speed protection and safe distance between trains. [20] and [22] apply the model-based testing on radio block systems for trains. [19] discuss the battery control system of train that manages the power source of the train system.

Table 6 Identified domains of model-based testing for software safety

Domain	Identified subcategory	Studies
Automotive	Car alarm system	[12], [17], [23], [30]
	Cruise control	[13], [30], [36]
	Car door controlling	[9]
	Car application system	[1]
	Control system	[3], [36]
	Landing gear system	[27], [36]
	Turn indicator	[31]
	Window controller	[30]
	ASR/ABS system	[32]
	Automotive operating system	[26]
	Electronic stability control	[36]
	Performance traction	[36]
	Electronic throttle	[36]
	Heating, ventilation, air conditioning	[36]
	Active safety control	[36]
Railway	Interlocking system	[8], [16], [29]
	Control system	[2], [4], [7], [21]
	Onboard system	[5]
	Radio block system	[20], [22]
	Battery control system	[19]
	Automatic train protection	[33]
	Programmable logic controller	[25]
Robotics	Autonomous mobile robots	[15]
	Vacuum cleaner	[14]
	Robot arm	[30]
Nuclear	Safety injection system	[10], [13]
Aerospace	Launch system	[28]
	Landing symbology function	[35]
Automation	Modular production system	[6]
Avionics	Helicopter system	[24]
	Autopilot system	[34]
Medical	Infusion pump	[11]
Power consumption	Power state machine	[18]

The domain *Robotics* includes three subcategories that are autonomous mobile robots, vacuum cleaner, and robot arm. [15] apply model-based testing on autonomous mobile robot which behaves like a human and make decisions on their own or interact with humans. In [14], vacuum cleaner robot is used to verify proposed model-based testing approach. The robot is able to create a map of its placed environment, clean the room, and avoid collision with living beings. [30] apply model-based testing approach on robot arms which can be controlled by a joystick.

In the domain *Nuclear*, [10] and [13] apply model-based testing on safety injection systems that injects water into the reactor pressure vessel automatically. In the domain *Aerospace*, [28] apply the proposed approach on launch system helps to initialize, fire, prepare for flying, and launch the system. [35] use landing functionality to illustrate their approach. For the domain *Avionics*, [24] apply the model-based testing on hypothetical helicopter system. In [34], the authors use an autopilot system to illustrate their approach. In the domain *Automation*, [6] apply the model-based testing on modular

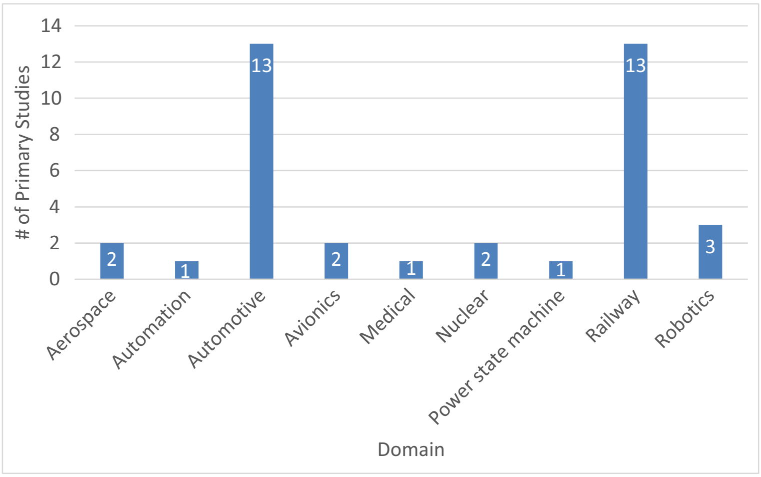

Fig. 4 Domain distribution of the studies

production system. In the domain *Medical*, [11] demonstrate the proposed solution approach for model-based testing on software which is developed for infusion pumps. In the final category *Power Consumption*, [18] illustrate the proposed methodology by using power state machine component which is used for power management in embedded systems. As seen in Table 6, [13] appears in both *Automotive* and *Nuclear* domains.

Based on Table 6, the approaches for model-based testing for software safety are applied to different types of domains. Additionally, it can be observed that *Automotive* and *Railway* domains are dominant in the selected primary studies.

4.4 RQ.2–existing research directions within MBT for software safety

RQ.2: What are the existing research directions within model-based testing for software safety?

With this research question, we aim to identify research directions within model-based testing for software safety. As defined in Section 3.2, we divide this research question into three sub-questions. The first sub-question aims to explain the motivation for adopting model-based testing for software safety, the second sub-question aims to present existing solution approaches, and the third sub-question aims to report identified research challenges.

4.4.1 RQ.2.1–motivation for adopting MBT

RQ.2.1: What is the motivation for adopting model-based testing for software safety?

Here, we aimed to identify the main reasons for applying model-based software testing for software safety in the reviewed primary studies. Based on the result of the data extraction process, we could derive the following reasons from the selected primary studies:

- *Reducing cost and development time*

Software testing has to be carried out carefully to ensure that a test coverage can detect the relevant faults. Unfortunately, as we have stated before, manual testing is a time-consuming process that becomes soon infeasible with the increasing size and complexity of the software. Also, in case of changes to the software regression, testing needs to be carried out to ensure that no faults have been introduced. Similar to MBT in general, an important motivation for MBT for safety is indeed the reduction of cost and development time. [3], [11], [12], [16], [17], [20], [21], [22], [23], [26], [30], [31] explicitly describe the reduction of cost and development time as the reasons for adopting MBT.

- *Improving the testing coverage*

Another main reason for adopting MBT is testing coverage which is the measurement of software testing how many lines/blocks/functions of code is tested. It describes how much of the code is exercised by running the tests. As the safety critical systems are growing, it is difficult to achieve high test coverage and complete testing by using conventional testing methods such as manual testing and random testing. [2], [3], [6], [10], [13], [18], [24], and [25] discuss how to achieve high testing coverage using MBT.

- *Improving the testing efficiency and quality*

The third main reason to apply MBT is increasing testing efficiency. [5], [12], [15], [16], [19], [26], [27], [28], [29], and [32] discuss how to increase testing efficiency and quality in their work. In the test case generation process, beside the generation of relevant test cases, redundant and irrelevant test cases may be generated. [5] indicates that in manual test case generation, most of the generated test cases cannot be reused and manual test case generation leads to repeated works when the configuration is changed. [16] discusses difficulty of quality evaluation of manually generated test cases regarding efficiency and redundancy. [15] point out that when test cases are generated in an ad hoc manner, they are described on a very low technical level of abstraction. [12] discuss the testing a software product line. The study indicates that testing every single product configuration of a software product line individually using common testing methods is not acceptable for large software product lines. Additionally, the study points out that in order to achieve efficient testing, small test suite must be generated to cover all test cases in the software product line. [19] focus on testing of functional block diagrams that represent component model of the safety-critical systems. In [19], the authors state that program testing of functional block diagrams mostly relies on manual testing or simulation methods which are inefficient way of testing. In [26], they discuss the testing of system regarding both functionalities and safety properties of the system in more effective way than the conventional testing. [27] focus on the lack of easy-use tool sets, guidance, and methodology to support development process. [28] point out the testing of system in terms of both functional and fail-safe behaviors. In [29], they discuss to make the testing activity more effective and facilitate incremental test-case generation. [32] focus on the techniques to develop automated quality assurances.

- *Increasing fault detection*

The last main reason to apply MBT is increasing the fault detection. In [1], [9], [13], [18], and [22] enhancing fault detection is discussed. [1] indicate that because of the increasing occurrence of failures in embedded systems in automotive domain, number of recalled cars increase. Therefore, testing is important to detect faults. [9] point that failures can be discovered by applying model-based testing. [13] indicate that written test cases can be used to check the implementation of software for faults. The fault detection capability can be improved by creating suitable test cases. Therefore, by applying testing process, fault detection can be improved. [18] indicate that designing system-on-a-chip has many challenges and automatic test case generation is necessary to find faults in these designs. [22] focus to minimize chances of failures.

In Fig. 5, we present the number of studies which include the mentioned four main reasons for applying MBT for safety. As shown in the Fig. 5, one primary study can discuss more than one main reason.

Apart from these main reasons, there are also minor reasons mentioned in reviewed studies. One minor reason is the need for particular set of models for testing. [7] consider the systems that are built up with components connected a network-like structure. It indicates that in these systems, each instance needs its own set of models for testing. Another minor reason is solving the state space explosion problem in automated verification techniques. [8] point that in model checking approach, when too many objects are taken into account, state space explosion problem arises. The other minor reason is reducing the complexity of process of creating test models. [21] consider this problem and create test specification patterns to help to solve this problem. Another minor reason is supporting the debugging process with model-based testing. [23] point to this issue and they aim to speed up the debugging process. The other minor reason is pointed out by [34] which aims to facilitate the application of model-based testing in avionics domain.

4.4.2 RQ.2.2–proposed solutions for MBT for safety

In this section, firstly we provide short overview of the selected 36 studies and then we present the results extracted from 36 selected primary studies in order to answer the research questions.

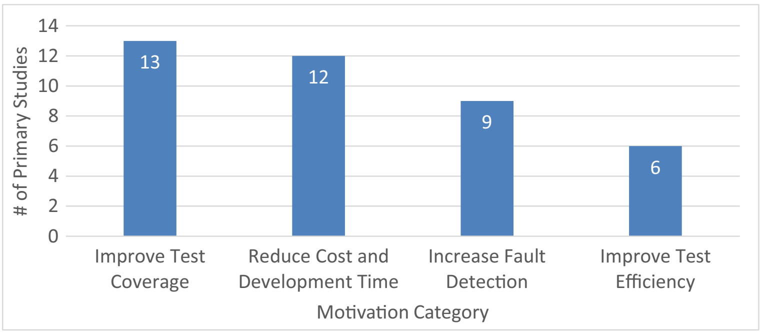

Fig. 5 Motivation for selected studies

As explained in the search scope, we include the studies published over the period of 1992 and August 2015. After the selection process, we have 36 selected papers over the years between 2005 and 2015. Figure 6 shows the year-wise distribution of the selected 36 primary studies along with the publication venues of the selected studies.

We present the overview of the selected primary studies according to publication channel in Table 7. The table includes the publication sources, publication channels, types of studies, and number of studies. According to the table, we can observe that the selected primary studies are published in highly ranked publication sources such as IEEE, Science Direct, and Springer. The journal *International Journal on Software Tools for Technology Transfer* is one of the remarkable publication channels that discusses all aspects of tools that aid in development of computer systems. The other significant publication channel is "Electronic Notes in Theoretical Computer Science" that provides rapid publication of conference proceedings, lecture notes, thematic monographs and similar publications of interest to the theoretical computer science and mathematics communities. The publication channels "Formal methods for industrial critical systems", *Information and Software Technology*, "Tests and proofs," and "Software Testing, Verification and Validation Workshops" are also distinguished publication channels. "Formal methods for industrial critical systems" provides a forum for researchers who are interested in the development and application of formal methods in industry. *Information and Software Technology* focuses on research and experience that contributes to the improvement of software development practices. "Tests and proofs" provides a forum for the cross-fertilization of ideas and approaches from the formal verification community and the testing community, abandoning earlier dogmatic views on the incompatibility of proving and testing. "Software Testing, Verification and Validation Workshops" focuses on research in all areas related to software quality.

RQ.2.2: What are the proposed solutions in model-based testing for software safety?

In order to analyze identified studies, we aimed to present the details for the selected 36 studies based on the data extraction process. As described in Section 3.6, we create a

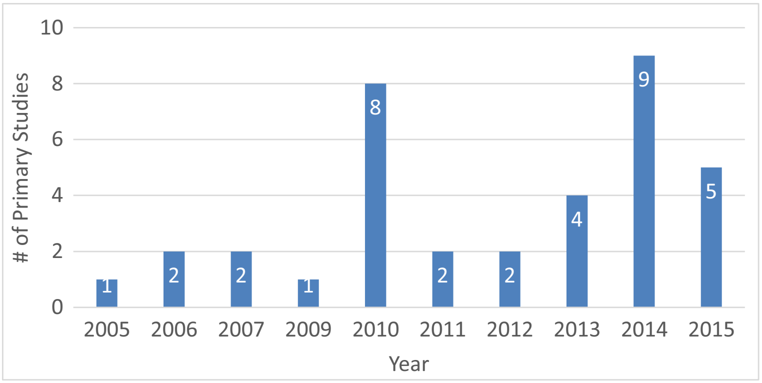

Fig. 6 Year-wise distribution of the studies

data extraction form considering the defined research questions and the model-based testing process shown in Fig. 1. Based on the results of data extraction process, we construct Table 8 to provide more detailed analysis of identified 36 studies.

As explained in 2.1, model-based testing process starts with constructing of the test models. In this step, system models are extracted from requirements or specification documents. In order to analyze this step, we extract the information about *existence of safety model*, *requirement specification language*, *model specification language*, and *method for model generation from requirements*.

In order to test safety properties of the software, it is quite important to create safety models from requirements. Only [2], [4], [6] and [28] (11% of the primary studies) create the specific safety model in order to describe the safety properties/ functions of the system under test. Eighty-nine percent of the primary studies do not describe the safety model in their studies.

For the requirement specification language, we define two categories: formal and informal. Eleven (31%) of the primary studies define the requirements formally. Fifteen (42%) of the primary studies define the requirements informally. Ten (28%) of the primary studies do not specify the requirements.

Table 7 Distribution of the studies over publication channel

Publication channel	Publication source	Type	Number of studies
International Journal on Software Tools for Technology Transfer	Springer	Journal	4
Electronic Notes in Theoretical Computer Science	Science Direct	Conference	3
Formal methods for industrial critical systems	Springer	Chapter	2
Information and Software Technology	Science Direct	Journal	2
Tests and proofs	Springer	Chapter	2
Software Testing, Verification and Validation Workshops (ICSTW)	IEEE	Conference	2
Software Testing, Verification and Validation (ICST)	IEEE	Conference	2
Agent and multi-agent systems. Technologies and applications	Springer	Chapter	1
Autonomous Decentralized Systems (ISADS)	IEEE	Conference	1
Computational Intelligence and Software Engineering (CiSE)	IEEE	Conference	1
Computer safety, reliability, and security	Springer	Chapter	1
e & i Elektrotechnik und Informationstechnik	Springer	Article	1
Formal methods for components and objects	Springer	Chapter	1
Formal Methods in System Design	Springer	Journal	1
High Level Design Validation and Test Workshop	IEEE	Conference	1
Information Technology and Applications	IEEE	Conference	1
Intelligent Solutions in Embedded Systems	IEEE	Conference	1
Intelligent Transportation Systems	IEEE	Conference	1
KI 2010: advances in artificial intelligence	Springer	Chapter	1
Model driven engineering languages and systems	Springer	Chapter	1
Model-based safety and assessment	Springer	Chapter	1
Software & Systems Modeling	Springer	Journal	1
Software Quality Journal	Springer	Journal	1
Model-Driven Engineering, Verification, and Validation (MoDeVVa)	IEEE	Conference	1
Engineering of Complex Computer Systems (ICECCS)	IEEE	Conference	1
Software Testing, Verification and Reliability	Wiley	Journal	1

Table 8 Data extraction for selected studies

Study	Reference	Requirement specification	Model specification	Generate models from requirements	Abstract test case generation	Type of generated test elements	Approach to generate test elements	Test selection criteria	Test case specification	Method to execute tests
[1]	Kandl et al. (2006)	Formal	Automata	Automatic	N/A	Test case, test data	Model checking	N/S	Informal	N/S
[2]	Yu and Xu (2010)	Informal	Automata	Manual	N/A	Test case, test sequence, test scenario, test script	N/S	N/S	N/S	Automatic
[3]	Fang et al. (2012)	Informal	DSL	Manual	N/A	Test case, test sequence	Model checking	Classification tree	N/S	Automatic
[4]	Yu et al. (2009)	Formal	Automata	Manual	N/A	Test case, test script, test data	Model checking	Temporal Logic Formulas	Informal	Automatic
[5]	Lv et al. (2013)	N/S	Automata	Automatic	N/A	Test case, test suite	Tool	Coverage criteria	N/S	N/S
[6]	Kloos et al. (2011)	Formal	Automata	Automatic	N/A	Test case	N/S	N/S	N/S	Automatic
[7]	Kloos and Eschbach (2010)	Formal	DSL	Automatic	N/A	Test model	DSL	N/S	N/S	N/S
[8]	Kollmann and Hon (2007)	Informal	UML	Manual	N/A	Test case	Multi-object checking	N/S	N/S	N/S
[9]	Lochau and Goltz (2010)	Informal	Automata	Manual	N/A	Test case	Graph algorithm	Adequacy criteria	Formal	N/S
[10]	Tseng and Fan (2013)	Informal	UML	Manual	N/A	Test case, test scenario	Algorithm	N/S	Formal	N/S
[11]	Auguston et al. (2006)	Informal	DSL	Manual	N/A	Test case	Tool	N/S	N/S	Automatic
[12]	Cichos et al. (2011)	Informal	Automata	Manual	N/A	Test suite	Tool	N/S	Formal	Automatic

Table 8 (continued)

Study	Reference	Requirement specification	Model specification	Generate models from requirements	Abstract test case generation	Type of generated test elements	Approach to generate test elements	Test selection criteria	Test case specification	Method to execute tests
[13]	Gargantini (2007)	N/S	Automata	N/S	N/A	Test sequence	Tool	Adequacy criteria	N/S	N/S
[14]	Micskei et al. (2012)	Formal	UML	Manual	Applied	Test data, test oracle	Model transformation	N/S	N/S	Automatic
[15]	Proetzsch et al. (2010)	N/S	Graph	Manual	N/A	Test case	Tool	N/S	Formal	Automatic
[16]	Herzner et al. (2010)	Formal	OOAS	Automatic	Applied	Test case	Tool	N/S	Formal	N/S
[17]	Krenn et al. (2010)	Formal	OOAS	Automatic	N/A	Test model	Tool	N/S	N/S	N/S
[18]	Mathaikutty et al. (2007)	Informal	DSL	Manual	N/A	Test case, test data	Tool	Coverage criteria	Formal	Automatic
[19]	(E. P. (Enoiu et al. 2013))	N/S	Automata	Automatic	N/A	Test suite	Tool	Adequacy criteria	N/S	Automatic
[20]	Zheng et al. (2014)	N/S	Graph	Manual	N/A	Test case, test sequence	Algorithm	Algorithm	Formal	Automatic
[21]	Gentile et al. (2014)	Informal	UML	Manual	N/A	Test sequence	Model checking	N/S	N/S	N/S
[22]	Marrone et al. (2014)	Formal	UML	Manual	N/A	Test case	Model checking	N/S	Formal	N/S
[23]	Aichernig et al. (2014)	N/S	Automata	Manual	N/A	Test case	Tool	N/S	N/S	N/S
[24]	Wilkinson et al. (2014)	Informal	DSL	Manual	N/A	Test case	Tool	N/S	N/S	N/S
[25]	(E. P. (Enoiu et al. 2014))	N/S	Automata	Manual	N/A	Test case	Tool	Logic coverage	N/S	Automatic

Table 8 (continued)

Study	Reference	Requirement specification	Model specification	Generate models from requirements	Abstract test case generation	Type of generated test elements	Approach to generate test elements	Test selection criteria	Test case specification	Method to execute tests
[26]	Choi and Byun (2017)	N/S	DSL	N/S	N/A	Test sequence	Tool	N/S	N/S	Automatic
[27]	Arcaini et al. (2017)	Informal	Automata	Manual	N/A	Test case	Framework	N/S	Formal	Automatic
[28]	Gario et al. (2015)	Formal	Automata	Automatic	N/A	Test model	Algorithm	N/S	N/S	N/S
[29]	Aichernig et al. (2015)	Informal	DSL	Manual	N/A	Test data	Algorithm	N/S	N/S	N/S
[30]	Schrammel et al. (2016)	N/S	DSL	Manual	N/A	Test case	Tool	N/S	N/S	Automatic
[31]	Kim et al. (2015)	Informal	Automata	Manual	N/A	Test case	Tool	N/S	N/S	N/S
[32]	Herber and Glesner (2015)	Formal	Automata	Automatic	N/A	Test case	Tool	N/S	N/S	N/S
[33]	Grasso et al. (2010)	Informal	Stateflow	Manual	N/A	Test scenario	Tool	N/S	N/S	Automatic
[34]	Stallbaum and Rzepka (2011)	Informal	UML	Manual	N/A	Test model	Manual	N/S	N/S	N/S
[35]	Samih et al. (2014)	Formal	DSL	Automatic	N/A	Test suite	Tool	N/S	N/S	N/S
[36]	Mohalik et al. (2014)	N/S	Stateflow	N/S	N/A	Test suite	Tool	N/S	Formal	Automatic

OOAS object-oriented action system, DSL Domain Specific Language, UML Unified Modeling Language, *N/A* not applied, *N/S* not specified

Model generation from requirements can be performed manually or automatically. In 36 selected studies, we identify that ten (28%) of the primary studies generate models from requirements automatically. Twenty-three (64%) of the reviewed primary studies generate models manually. Three (8%) of the studies do not explain their model generation method explicitly.

For the model specification language, the reviewed primary studies used various different specification languages. In Fig. 7, we present all extracted methods from the 36 selected primary studies. In 15 (42%) of the primary studies ([1], [2], [4], [5], [6], [9], [12], [13], [19], [23], [25], [27], [28], [31], [32]), automata are used as model specification language. Automata are a useful model for various different kinds of hardware and software (Hopcroft et al. 2001). In [1], NuSMV NuSMV Language (NuSMV Language n.d.) which is designed for model checking is used to declare models. In [2] and [6], finite state machine is used as model specification language. In [28], model is defined as communicating extended finite state machine which is based on finite state machines. In [5], [19], [23], [25], [31], and [32], models are defined as timed automata. In [9], models are described by using StateFlow which has been adopted from StateChart and allows hierarchical modeling of discrete behaviors consisting of parallel and exclusive decompositions, which makes it challenging to capture and translate into formal models (Li and Kumar 2011). In [12], deterministic state machine is used to model products in a software product line. In [13] and [27], models are defined as abstract state machine.

In [3], [7], [11], [18], [24], [26], [29], [30], [35] (25% of the primary studies), models are defined using domain-specific languages which are designed to express statements in particular application domain. The verification language Promela is used in [3] as model specification language. In [7], they define a domain-specific language (DSL) to build a system model by defining composition operators and atomic components in the system. Attributed event grammar is used in [11]. Esterel language which is used for the development of complex systems is used as model specification language in [18]. In [24], they express Event-B models to create formal model of the system. [26] introduces a constraint specification language OSEK_CLS for modeling the system. In Kitchenham and [29], the authors formalize the requirements using a domain-specific language and use them as models. In [30], Linear Temporal Logic

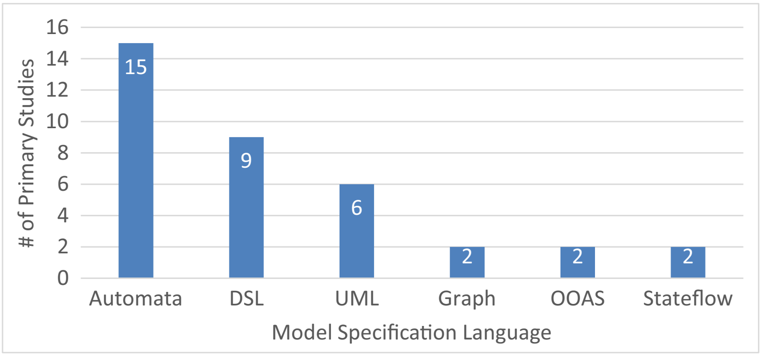

Fig. 7 Model specification language

(LTL) is used to indicate safety concerns in the system. In study of [35], usage model variants models from MaTeLo Tool (n.d.)) ("MaTeLo Tool") are used to model the system.

In six (17%) of the primary studies, Unified Modeling Language (UML) is used to construct models. In [10] and [14], UML sequence diagram is used to define test models. UML state diagram is used as a model specification language in the study by [21]. UML profiling is used for modeling in [22]. In [7], UML-based railway interlocking (RI) models are used to define test models. UML-based RI includes the infrastructure objects and UML to model the system behavior. [34] use UML profiles to define test models to support DO-178B certification.

[33] and [36] use Simulink/Stateflow models as test model. [20] define the models by using Colored Petri Net (Jensen 1987) graphs. In [15], models are defined as product graphs. In [16] and [17], Object-Oriented Action System (OOAS) is used as model specification language. OOAS is used for formalism of parallel and distributed systems.

The second step for model-based testing is generating abstract test suites as shown in Fig. 1. We analyze the primary studies in terms of application of this step. Most of the studies do not apply this step in their approaches. Only two (6%) of the studies, [14] and [16], applied the abstract test suite generation process before generating concrete test suites.

For test case generation step, only four of the primary studies, [7], [17], [28] and [34] do not conduct the test case generation step. They perform only model construction step. Therefore, there is no extracted data for test case generation step regarding these studies. Additionally, in some reviewed studies, test data (inputs and outputs), test sequences, test scenarios, test oracles, and test scripts are generated beside of the test cases.

In Fig. 8, we present the generated type of test elements along with the number of studies. The reviewed studies, except studies [7], [12], [13], [14], [17], [19], [21], [26], [28], [29], [33], [34], [35] and [36] generate test cases. [2], [3], [13], [20], [21], and [26] generate test sequence which is the set order of steps and actions comprising a test or test run. [1], [4], [14], [18] and [29] generate test data which is used for testing of system. [2], [10] and [33] generate test scenario that represents the set of actions in order to test the functionality of the system. [2] and [4] generate test scripts which are a set of instructions in order to test system functions correctness. [14] generate test oracle which is a mechanism that decides whether system has passed or failed a test. [5], [12] [19], [35] and [36] generate test suites.

In order to generate types of test elements (test case, test script, etc.), reviewed studies propose various types of solution approaches. In Fig. 9, we present the proposed solution approaches for generating test elements.

As seen from the Fig. 9, 17 (47%) of the primary studies use existing model-based testing tools. [5] uses CoVeR tool (Hessel and Pettersson 2007a) to generate test cases automatically based on timed automata theory. CoVeR is a model-based testing tool that allows its users to automatically generate test suites from timed automata specifications of real-time systems. [11] generate test cases and test scripts by using attributed event grammar-based (AEG-based) generator. It is used for automation of random event trace generation in order to generate desired test cases and test scripts. [12] use a model-based testing tool Azmun as test case generator that is based on the model checker NuSMV (NuSMV Language n.d.) to generate test cases of products in software product line. [13] use a tool ATGT (ATGT Tool n.d.). [15] use JUMBL (J

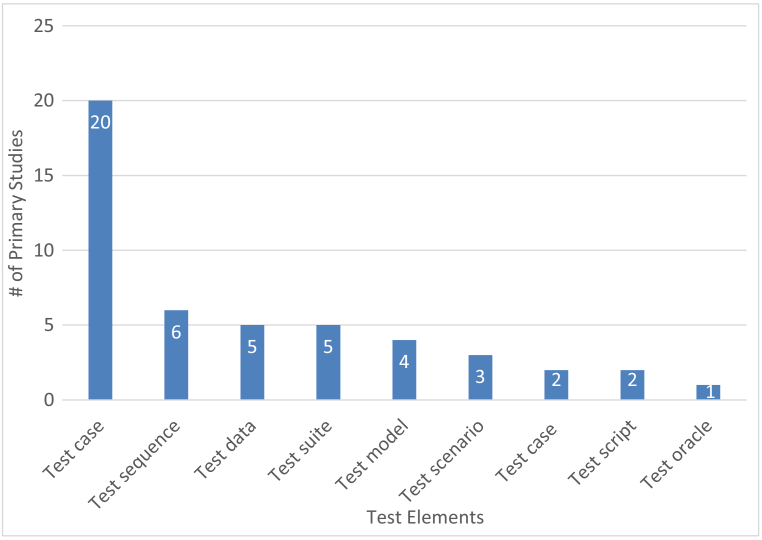

Fig. 8 Generated type of test elements

Usage Model Builder Library) tool (Prowell 2003) which is a model-based testing tool for statistical testing in order to generate test cases and test scripts. [17] use Argos and Ulysses (Aichernig et al. 2011) tools to generate OOAS models from UML diagrams. [18] use Esterel and TestSpec Generator in order to generate executable test suites from abstract test suites. [19], [25] and [31] use the UPPAAL tool based on model checking to generate test cases from models. [23] use the model-based testing mutation tool MoMuT::TA for timed automata. In [30], they use test case chain generator CHAINCOVER tool. [32] use Verista tool to generate test cases. [16] uses VIATRA tool to generate OOAS models from UML diagrams. [33] implement their own tools Test Observator and Test Integrator to generate test scenarios. [35] use tool MaTeLo to generate test suites. [36] implement their own tool AutoMOTGen to generate test suites from Simulink/Stateflow models.

The second most used solution approach is model checking. Seven (19%) of the reviewed primary studies used model checking to generate test elements. Model checking is a technique used for formal verification of the system automatically. The main purpose of the model checking is to verify a formal property given as a logical formula on a system model. Model checkers are formal verification tools which are capable of providing counter examples to violated properties (Fraser et al. 2009). [1] use SAL and NuSMV model checkers to generate test case and test data. SAL ("Symbolic Analysis Laboratory Title (Symbolic Analysis Laboratory Title n.d.) is a framework which is used for model checking of transition systems. NuSMV Language (NuSMV Language n.d.) is a model checker based on binary decision diagrams. It is designed to be an open architecture for model checking. [26] uses the model checker NuSVM in order to generate test sequences. In [3], they aim to find both test cases and execution sequence by using model checking techniques. [4], [21], and [22] use the SPIN Spin–Formal Verification (Spin–formal verification n.d.) model checker tool in order to

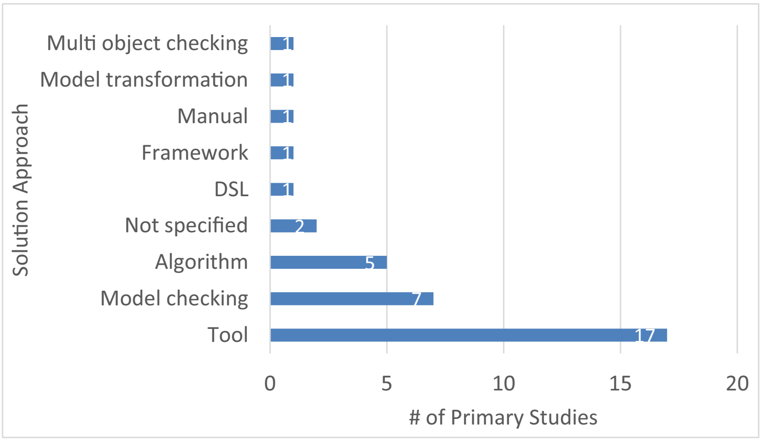

Fig. 9 Solution approaches for generating test elements

generate test cases and test scripts. SPIN is a general tool for verifying the correctness of distributed software models automatically. The model checker Rodin is used to generate test cases in [24].

In five (14%) of the reviewed studies, algorithm is used to generate test cases. In [9], they use path-finding algorithm on a graph to generate test case. [20] use all the paths covered optimally graph algorithm to generate test cases. [10] define a new algorithm which generates test cases by extracting the data from the tagged preliminary safety analysis report (PSAR). The extracted data generate the sequence diagram to product test information. In [28], they apply the algorithm defined in Hessel and Pettersson (2007b). [29] define a new algorithm to generate test cases incrementally.

[8] uses a multi-object checking in order to generate test cases. In model checking techniques, if too many objects are taken into account, state space explosion problem arises. Therefore, they use multi-object checking which outwits the state space explosion problem by checking one object at a time. [14] generate test data and test oracles by using model transformations by conforming model instances to the metamodel. [7] use a DSL to define test models. [27] use the ASMETA framework to generate test cases. [2] and [6] do not specify their methods used to generate test elements.

Next step for the model-based testing is definition of test selection criteria. For this step, most (75%) of the primary studies are not define the criteria for test selection. Only nine (25%) of the primary studies, [3] and [4] define the criteria. In order to define the criteria, [3] use classification tree, [4] use temporal logic as test selection criteria. The studies [5], [9], [13], [18], [19], [20], and [25] define their own criteria.

For the test case specification step, most (66%) of the primary studies don't specify their test case specification language. Ten (28%) of the reviewed studies [9], [10], [12], [15], [16], [18], [20], [22], [27] and [36] define test cases formally. Two (6%) of the primary studies, [1] and [4], use an informal language to describe test cases.

Test execution can be done manually or automatically. For this step, 17 (47%) of the primary studies [2], [3], [4], [6], [11], [12], [14], [15], [18], [19], [20], [25], [26], [27], [30], [33] and [36] execute tests automatically. Nineteen of the primary studies do not state explicitly whether they run the tests or do not.

With this research question, we also extracted information about contribution provided by the reviewed primary studies and present them in Fig. 10. Twenty-five (69%) of the primary studies propose a method in order to conduct model-based testing for software safety. Seven (19%) of the primary studies implement a framework, only 4 of the reviewed primary studies a tool to test software safety by using model-based techniques.

4.4.3 Summary of the selected primary studies

Below we provide a short summary of each primary study.

- *Study* [*1*]: In this work, the authors present the requirements in temporal logic formulas. An automaton model is generated in NuSVM from the c-source code automatically. They generate the test cases from the automaton model and requirement specification using the model checkers SAL and NuSVM by producing counterexamples. The approach is illustrated using a case study from automotive domain.
- *Study* [*2*]: In this study, the authors provide an automaton model for safety properties. The safety model is generated from automaton model. Test case and test script generation are performed based on the safety model. They provide a framework for the testing process. The proposed approach is validated using an industrial case from railway domain.
- *Study* [*3*]: The authors propose a method for model-based testing of AUTOSAR multi-core RTOS. Firstly, they construct an abstract model to describe requirements. From this model, they generate concrete model in the Promela language with system configuration. Then, from this formal model, they generate the test cases by model checking. They provide a classification tree for test selection. Additionally, they provide a method for bug analysis. The proposed approach is illustrated using an experiment from automotive domain.
- *Study* [*4*]: In this study, the authors propose a framework for generating test cases from a safety model. Firstly, they model the system using finite state machine (FSM). The FSM models are translated into Promela models. Each test requirement is formulated as temporal logic expression. In addition to these models, Markov chain model is used to describe the states of the system. Test case generation is performed by SPIN tool with model checking techniques using the constructed models. They illustrate the proposed framework on an industrial case from railway domain.
- *Study* [*5*]: In this study, the authors propose a new algorithm for test case generation to support the testing of onboard systems. Firstly, they produce the network timed automata model from interaction model of system using the UPPAAL tool. Then,

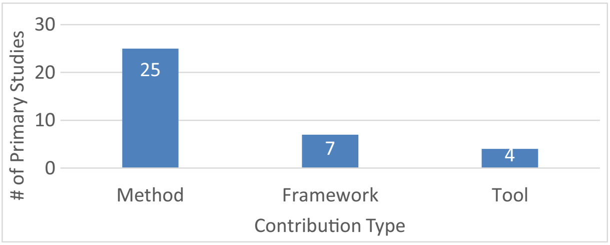

Fig. 10 Contribution type

they generate the test cases from network timed automata model using the CoVeR model-based testing tool. The proposed approach is illustrated using a case study from railway domain.

- *Study* [*6*]): In this work, the authors propose a risk-based testing method using the information from Fault Tree Analysis (FTA). They generate test cases based on the risk given in FTA. They use the event set notion and transform the event set into state machine as test model. They mainly focus on generating the test model from FTA events. The proposed approach is illustrated by using an automation system.
- *Study* [*7*]: The authors focus on generating test model for the instances in the system. Firstly, they identify the components and composition operators in the system. Then, they describe the behavior of components using the Mealy machines (type of finite state machine) and behavior of composition operators using π-calculus. They define a DSL which uses the components and composition operators to build a system model from domain description. The proposed approach is illustrated by using a case study from railway domain.
- *Study* [*8*]: In this paper, the authors focus on the state space explosion problem in model checking process. They propose a multi-object checking approach for generating scenarios in order to solve state space problem. Firstly, they define the UML models of the system by using UML-based railway interlockings. Then, they propose an approach for generating counterexamples with multi-object checking. From the UML-based RI models, they generate the counterexamples using the multi-object checking. Based on the counterexamples, they generate test cases with multi-object checking method. The approach is illustrated on a case study from railway domain.
- *Study* [*9*]: In this study, the authors propose a model-based test case generation approach particularly aim feature interaction analysis. Firstly, they define the functional architecture and behavioral specification to describe system specification model. Functional architecture defines the components, sensors, actuator hardware devices and values, such as signals and shared variables in the system. Behavioral specification describes the behavior of the system by using the STATEFLOW automata. In order to generate test cases, the STATEFLOW diagrams are transformed into flow graphs. They generate the test cases from the flow graphs. The approach is illustrated by using a case study from automotive domain.
- *Study* [*10*]: In this paper, the authors propose a systematic method for test case generation based on a PSAR. The report is written in natural language which specifies the user's needs. They convert the PSAR into an explicit system model for scenario-based test case generation. Then, they design ontology that represents the set of concepts and their relations with in a domain. They construct the SRP (Standard Review Plan)-based ontology in XML that will be used to tag PSAR. Sequence diagram is generated for combining and generating different scenario test cases from the tagged PSAR. The test cases are generated from the sequence diagrams and their variations. They illustrate the proposed method using a case study from nuclear domain.
- *Study* [*11*]: In this paper, the authors present an approach for automatic scenario generation from environment behavior models of the system. The authors define an environmental behavior model rather than system behavior model. The environmental behavior model focuses on the productive aspects of the behavior. They model the environmental behavior of system as event trace. Then, they use the AEG tool for generating AEG model from environment model. The test generator takes the AEG and derives a random event trace

from it and generates a test drive in C. They illustrate the proposed approach using an experiment from medical domain.

- *Study* [*12*]: In this work, the authors provide an approach for test suite generation for testing of software product lines (SPLs). They define their test model as state machines. For each product in the SPL, they build a test model called as 100% test model. By combining these models, they build a super model called as 150% test model for SPL. Additionally, they define the test goals for test case selection. Then, they propose an algorithm to generate test cases from the 150% test models using the test goals. They use the Azmun framework as a test case generator. The proposed method is illustrated on a case study from automotive domain.
- *Study* [*13*]: In this study, the authors focus on fault detection. They classify the faults and select most studied classes of faults in the literature. They use the abstract state machine (ASM) as a test model. Based on the ASM and fault class, they generate the test predicates that describe the test conditions. From the ASM specification, SPIN model checker generates the counterexamples with model checking. Based on the counterexamples and test predicates, the test suite is generated. They illustrate their approach using one case study from automotive domain and one from nuclear domain.
- *Study* [*14*]: In this study, the authors focus on automatically generating test data representing complex situations and evaluating test traces. In this case, it is important to have a method providing a method to generate both test data and test oracle. Firstly, the authors define the context model and scenarios in the system. Context model is a metamodel of the system and explains the elements and their relations. The scenarios are presented in UML sequence diagram of the system. Based on the context model and UML sequence diagrams, they generate test data. For the test oracles, they present requirements as graphical scenarios in the form of extended UML sequence diagrams which express events/messages received and actions/messages sent by the system under test. Based on these action models, they generate test oracles. The proposed approach is illustrated on a case study from robotics domain.
- *Study* [*15*]: In this work, the authors construct the test model as transition system that includes all possible inputs and corresponding expected outputs. In addition, they define a DSL for expressing transition systems. They use JUMBL tool for test case generation. The proposed approach is illustrated by using an experiment from robotics domain.
- *Study* [*16*]: In this paper, the authors define the UML class diagrams and state diagrams to express the requirements. In order to express the rules that define the system behavior, they use the OCL. They generate the OOAS models from UML diagrams using VIATRA tool. OOAS consists of a finite set of variables representing the state of system and a finite set of actions that act upon the variables. They generate the mutants of the OOAS models. For every OOAS model and its mutants, they generate input/output-labeled transition system (IOLTS) as abstract test cases. IOLTS describes the states and transition relations between these states. The abstract test cases are converted to EPS (Elektra Periphery Simulator) scripts that present concrete test cases. They illustrate the proposed method on a case study from railway domain.
- *Study* [*17*]: In this study, the authors present an approach for generate OOAS model as test model from UML class and state diagrams. They define a set of rules for transformation UML diagrams into OOAS model. They implement a tool for transformation.

Additionally, they use the Argos tool which converts OOAS model to an action system that is the input for their test-case generator Ulysses. The proposed approach is illustrated by an industrial case from automotive domain.

- *Study* [*18*]: In this paper, the authors derive the functional model from the requirement specification in a language called ESTEREL. They also build verification model in PSL (property specification language). They annotated these models according to defined code coverage metrics and they produce structural and conformance models. From these models, tests are generated by esVerify tool by generating counterexamples. The generated tests are not executable. They are transformed into executable SystemC tests using the TestSpec generator. The proposed method is illustrated by using a power state machine.
- *Study* [*19*]: In this paper, the authors propose an approach to transform functional block diagram (FBD) into timed automata model. Programmable logic controllers widely used in avionics and railway domains. FBD is a programming language for PCLs. They use a UPPAAL model-checker to generate test cases from timed automata model. The proposed method is illustrated using an industrial case from railway domain.
- *Study* [*20*]: In this work, the authors, firstly, build the colored Petri Net (CPN) model based on the system requirement specification. Based on the CPN model, XML file and reachable graph of the CPN model are obtained. They propose an algorithm All Paths Covered Optimal (APCO) to generate test cases as XML. From the XML test cases, they apply the APCO algorithm to obtain set of test subsequences. The set of XML test sequences is generated using the Sequence Priority Selected (SPS) algorithm. The proposed method is illustrated using an industrial case from railway domain.
- *Study* [*21*]: In this paper, the authors propose test specification patterns (TSP) that are constructed based on the requirements and they provide a set of guidelines for test specification. They propose a Verification&Validation UML profile to capture system and requirement features. The test specification patterns are expressed in UML statechart diagrams annotated with V&V UML profile. As a second step, TSPs are transformed into Promela models that specify behaviors should not happen. Using the SPIN model-checking tool, Promela code and test sequences are generated from Promela models. The authors illustrated the proposed work on the industrial case study in railway domain.
- *Study* [*22*]: In this work, the authors start with building a high-level model of the system includes the information for Validation & Verification (V&V) purposes expressing the structure and behavior of the system as well as modeling its requirements. For this purpose, they define the UML V&V profile. They develop a transformation that generates a Promela specification from the high-level UML model using Atlas Transformation Language (ATL). Using the SPIN model-checking tool, the test cases are generated from the Promela models. They use an industrial case study from railway domain to illustrate the proposed approach.
- *Study* [*23*]: In this paper, the authors propose model-based mutation testing. They define the models in terms of timed automata with inputs and outputs (TAIO). Then they apply the mutation on these TAIO models. After that, they run the existing test cases on the mutated models and select the mutants that are failed (killed mutants) to create new test cases. They use a tool MoMuT::TA to create minimal test cases for

the killed mutants. They exemplify the proposed approach on two short examples from automotive domain.

- *Study* [*24*]: In this paper, the authors use Event-B grammar to create formal model of the system. Event-B is a formal method to model and analyze the system. While defining Event-B models, they apply system-theoretic process analysis to discover potential hazards in the system. With this process, they introduce the safety constraints in Event-B models. They use a model checker ProB to generate abstract test cases from Event-B models. In order to show the applicability of their approach, they create a case study from avionics domain.
- *Study* [*25*]: In this work, the authors aim to generate test cases for FBD programs. Firstly, they transform the FBD programs to timed automata models. After that, they annotate the transformed models such that a condition describing a single test case can be formulated. They use model-checking tool UPPAAL to generate test cases from timed automata models. They develop a toolbox COMPLETETEST for the proposed approach. The proposed work is demonstrated with the industrial experiment from railway domain.
- *Study* [*26*]: In this paper, the authors introduce a constraint specification language OSEK_CLS to define usage constraints of automotive systems. They develop a tool which generates constraint modules and trap properties from constraint specs and system configurations. The constraint modules and trap properties combined with generic task models. Then, they used NuSVM model checker to generate tests from these models. They illustrate the proposed approach on automotive operating system.
- *Study* [*27*]: In this work, they develop a framework ASMETA as an Eclipse plug-in for their proposed approach. Firstly, they create an ASM model capturing the behavior of the system at a high-level of abstraction based on the informal requirements. The ASM models are an extension of finite state machines. They generate abstract test sequences from ASM models using ATGT tool in ASMETA framework. After this step, they generate JUnit tests based on these abstract test sequences. The proposed work is demonstrated using an industrial experiment from automotive domain.
- *Study* [*28*]: In this paper, the authors propose a testing approach based on behavioral and fault model of the system. They express the behavioral model as a communicating extended finite state machines (CEFSM) and the fault model as a fault tree. As a first step for their approach, they apply compatibility transformation using behavioral model and fault tree and construct transformed fault tree. They transform the transformed fault tree into Gate CEFSM (GCEFSM). For third step, they integrate the CEFSM and GCEFSM models into integrated CEFSM (ICEFSM). They develop a tool that provides these transformations. They use these models while applying fail-safe testing on a case study from aerospace domain.
- *Study* [*29*]: In this work, the authors formalize the requirements by defining a domain-specific language. They use the formalized requirements as models. They define an algorithm that uses SMT solver Z3 to generate test input sequences from these models. They illustrate their method on wheel loader case study and interlocking case study from railway domain.
- *Study* [*30*]: In this paper, they aim to discover a test case chain which is a single-test case that covers a set of multiple test goals and minimizes the test execution

time. They build a tool chain for the proposed method and define LTL properties to indicate safety concerns in the system. They generate the C code from SIMULINK models using the GENE-AUTO tool. Using the CHAINCOVER tool, they generate test cases from generated C code and defined LTL properties. They exemplify their tool chain on some experiments from automotive and robotics domain.

- *Study* [*31*]: In this work, the authors model the system as network of timed automata using the UPPAAL environment. Firstly, they analyze and verify the timed automata model using UPPAAL statistical model checking with respect to the properties formulated based on the system requirements. After then, they generate test cases from the verified model using the UPPAAL Yggdrasil tool to check that whether behavior meet by the model. In order to demonstrate their work, they use an industrial case study from automotive domain.
- *Study* [*32*]: In this paper, they construct a framework VeriSTA based on a model checking and conformance testing. They use the SystemC to design the system model. They develop a transformation engine (STATE) to transform SystemC designs into UPPAAL timed automata. They generate conformance tests from these UPPAAL timed automata models using the VeriSTA framework. They illustrate the proposed approach by conducting experiments on a case from automotive domain.
- *Study* [*33*]: In this paper, the authors combine the model-based testing approach with abstract interpretation method which is a static analysis method for inferring dynamic properties of the code and catch faulty states of the program without executing the code. They construct the Stateflow model from defined unit test requirements. From the Stateflow model, they generate C code using real-time workshop (RTW) embedded coder. They describe the unit tests in Simulink test suite and develop a tool Test Observer to translate unit test definitions into test suites for the generated C code. Test Observer registers the test execution during the simulation in terms of input/output Simulink time series and translates these time series into given input/expected output matrixes for the generated code. They also develop a tool Test Integrator which creates a main file merging the registered input/expected output and the model generated code. For each test case, Test Integrator runs the executable file and checks whether the output is same with the expected output. They experimented the proposed tools using a case study from railway domain.
- *Study* [*34*]: In this study, the authors propose a new UML profile which can be used as a test model during MBT to support DO-178B certification. In their work, firstly, they explain the importance of the DO-178B certification and identify eight essential requirements which have high impact on to support DO-178B certification. From the identified essential requirements, they define a conceptual model for proposed UML profile for DO-178B compliant test models. Then, they exemplify their proposed profile using a case study from avionics domain.
- *Study* [*35*]: In this work, the authors introduce an approach that adopts MBT process for product lines (PL). Firstly, they define their product line requirements using the tool IBM Rational DOORS. Then they import these requirements into the tool MaTeLo and create PL usage model. MaTeLo transforms the PL usage models into usage model variants for designed products. The MaTeLo Testor tool

generates the test suites from these derived usage model variants. Finally, they assess their approach using a case study from aerospace domain.

- *Study* [*36*]: In this paper, the authors propose a tool AutoMOTGen which generates tests from Simulink/Stateflow design models and test specifications. Firstly, they define Simulink/Stateflow design models and requirements and test specifications. Based on design models and specifications, AutoMOTGen generates test suites using model checking techniques. They illustrate their approach using several case studies from automotive domain.

RQ.2.3–Research challenges in MBT for software safety

RQ.2.3: What are the research challenges in model-based testing for software safety?

This research question is aimed to reveal the research challenges that are extracted from primary studies for further advancements. With respect to this question, we identified some research challenges that include problems in reviewed studies and future research directions.

- *Model-based testing for domain-specific applications*

All reviewed papers discuss model-based testing for particular application domains, such as automotive and railway. There seems to be a clear impact of the specific domain on the model-based testing process. The question here is whether we could provide a general-purpose MBT approach without considering a particular domain. For this purpose, the application domain semantics and its impact on the MBT process should be investigated.

- *What is the impact of the context on MBT? How to model context in/for MBT?*

Some of the reviewed papers indicate that existing standard test descriptions do not support the representation of changes in the context. For some domains such as autonomous systems, safety testing has some challenges due to some reasons: Firstly, the system behavior is highly context-aware. Additionally, context is complex and its specification could be large. Thirdly, changes in system behavior and context should be handled to capture the requirements. In order to solve these problems, study of [16] defines a context model and scenario-based behavior specification language. The context model captures domain knowledge about context of the system systematically. In regard to the system behavior, scenario-based behavior specification captures the behavior of system in case of a test context.

- *What are the required metrics for validating/evaluating the MBT elements including model, test case specification, test case* etc.*?*

As explained before, model-based testing consists of several steps. In each step, at least one element is produced to complete MBT process. However, after each generation of MBT elements, the quality of the generated element should be evaluated. In some reviewed studies, they propose a new metric or use existing metrics to assess the testing quality. In the study by [15], they define context-related coverage metric and scenario-related coverage metric in order to measure testing coverage. In reviewed studies, there is not stated/proposed metric to evaluate for other types of MBT elements.

- *How to compose models for generating test cases in MBT?*

Some of the systems are built up with components that connected a network-like structure. In the study by [7], these systems are discussed. Each instance of these systems requires its own set of models to generate test cases. However, creating a test model for each instance could be costly. Therefore, they propose a component-based solution to generate test models by using general information. They create test model components from requirement specification and they translate these models by using the domain-specific information.

- *How to define MBT for software product families?*

Software product line (SPL) is an engineering approach for the systematic software reuse in order to reduce the cost and development time and improve the software quality. Since every product needs its own configuration in large SPLs, SPL testing approaches are not able to test efficiently large SPLs. Additionally, testing each product in SPLs individually is a time-consuming process. For these reasons, [12] propose a new approach for testing of SLPs. They implement an algorithm which generates a set of test cases from complete test model that consist of all test models of an SPL as special cases. They generate test cases which satisfy the required coverage criteria. After the test case generation, they applied selection criteria on generated test cases in order to represent all subsets of product features in the SPL. In [35], the authors introduce an approach that adopts MBT process for PL. Firstly, they define their product line requirements using the tool IBM Rational DOORS. Then they import these requirements into the tool MaTeLo and create PL usage model. MaTeLo transforms the PL usage models into usage model variants for designed products. They use the MaTeLo Testor tool to generate the test suites from these derived usage model variants.

- *How to apply MBT for testing systemic behavior?*

In some reviewed studies, they use behavioral models for test case generation. The study by [7] focuses only on creating proper test models for the embedded control systems. In order to handle the complexity of these systems, they propose a component-based approach. They identify the candidate components that represent the behavior of system. They use Mealy machine (finite-state automata) in order to describe the behavior of the components. They define a DSL that describes components and operators to build a system model as a test model. The study by [9] describes the behavioral models of system by using Stateflow finite-state automata) models. In the study by [8], they used UML sequence diagrams to define their behavioral models. In [22], they define a UML V&V profile which contains all information for V&V purposes, describing the structure and behavior of the system under test.

- *How to integrate MBT with other V&V approaches?*

The main purpose of model checking is to verify a formal property given as a logical formula on a system model. Model checkers are formal verification tools that have capability of providing counterexamples to violated properties. In some reviewed studies ([3], [4]), model checking is used to interpret both counterexamples to find test cases and the test cases to find execution sequence. However, the study by [8] indicates that model-checking techniques suffer from state space explosion problem when the system has too many objects. Hence, they propose multi-object checking approach to handle the state space explosion problem.

- *How to define a generic test model to express safety properties/functionalities of the system?*

In reviewed studies, only four of the primary studies have specific safety model to use it test case generation process. Three of these papers define their safety properties using automata. One of them defines a DSL in order to specify safety model. Based on these results, none of the reviewed papers provide a generic approach to generate test model. However, in Thomas et al. (2008), the authors propose a UML profile on architectural level aim to provide a tool for formal verification and validation techniques such as model checking and runtime verification.

- *How to generalize the safety requirement specification in order to generate test models?*

Based on the data extraction results, only six of the primary studies express the requirements by using formal language. Two of these studies use temporal logic formulas to indicate the requirements. The other studies use fault tree and UML state diagrams. As a result, there is no proposed generic approach to express safety requirements.

- *How to handle evolution of the system models in the test models?*

As a nature of continuous engineering, the evolution of system models is essential to reflect changes and improvements in the system. With regard to updating and improving the system models, it is expected to have some revision on test models. Based on the data extraction results, none of the primary studies consider the evolution of the system. Therefore, none of them discuss how to reflect the changes/improvements on the system model to test models.

4.5 RQ.3: What is the strength of evidence of the study?

We design the third research question to address strength of evidence based on the selected primary studies. In the literature, there are several systems for grading the strength of the evidence. In this work, we used the definitions from the Grading of Recommendations Assessment, Development and Evaluation (GRADE) (Atkins et al. 2004) working group which is developed for grading the quality of evidence and strength of recommendations. GRADE approach specifies four grades of strength of evidence which is given in Table 9 (adopted from Atkins et al. 2004). The strength of evidence is determined by four key elements which are study design, study quality, consistency, and directness.

Regarding the study design, the GRADE approach gives higher grade to experiments than to observational studies. In this work, 10 (28%) of the selected primary studies are experimental type. Table 10 shows the average quality scores related to experimental studies. Thus, according to GRADE approach, our first categorization of the strength of evidence in this review from the perspective of study design is low.

With respect to quality of studies, in general, issues of bias, validity, and reliability are not addressed explicitly. Additionally, none of the selected primary studies got full score from our study quality assessment criterion. Twenty-two (61%) of the selected primary studies stated their findings clearly in terms of credibility, validity, and reliability. Besides, none of the selected primary studies discuss the negative findings clearly. Based on these findings, we can conclude that there are some limitations to the quality of the selected primary studies due to the low-quality scores.

Regarding the consistency which addresses the similarity of estimates of effects across studies, we realized that there are little differences among studies. Because of the results of the primary studies are presented both objectively and empirically, we did not conduct a sensitivity analysis by excluding studies which have poor quality. Since the outcomes of reviewed primary studies are not presented in comparable way and reporting protocols vary from study to study, evaluating the synthesis of quantitative results will be not feasible. This causes us to perform the data synthesis in a more qualitative or descriptive way which is not desired. Based on these findings, we can conclude that in general results are consistent.

Directness refers to the extent to which the people, interventions, and outcome measures are similar to those of interest. In this context, people refer to the subject of the study; intervention refers to the applied model-based testing approaches. With respect to the people, none of the selected primary studies used human subjects. Regarding the intervention, in the selected primary studies, various types of model-based testing approaches are used. With respect to the outcome measures, 13 (40%) of the primary studies performed in industrial settings. Based on these findings, the total evidence based on directness of the primary studies is low.

Combining the four key elements of study design, study quality, consistency, and directness for grading the strength of evidence, we found that the strength of evidence

Table 9 Definitions for grading the strength of evidence

Grade	Definition
High	Further research is very unlikely to change our confidence in the estimate of effect
Moderate	Further research is likely to have an important impact on our confidence in the estimate of effect and may change the estimate
Low	Further research is very likely to have an important impact on our confidence in the estimate of effect and is likely to change the estimate
Very low	Any estimate of effect is very uncertain

Table 10 Average quality scores of experimental studies

Experimental studies	[3], [13], [15], [19], [20], [23], [25], [27], [30], [32]
Number of studies	10
Mean quality score	8.2

in a low grade. This means that the estimate of effect that is based on the body of evidence from current research can be considered uncertain. Further research is required to gain a reliable estimate of effects of model-based testing for software safety.

4.6 Threats to validity

One of the main threats to validity of this systematic mapping study is the publication bias. The publication bias indicates the tendency of researchers to more likely publish positive results. In order to deal with this bias, we develop a research protocol and constructed research questions. After this, we define our search scope and search method clearly. Since we decide to search papers automatically, we constructed our search string according to the target of this systematic mapping study. Another important issue is the incompleteness which results in search bias. The risk of this threat highly depends on used keywords in search string. In order to reduce this risk, we use an iterative approach in keyword list construction process. In order to achieve the largest set of targeted search items, we perform some pilot searches on search engines of selected electronic databases by constructing a keyword list. When the keyword list was not able to find the targeted studies, new keywords were added to list or some keywords are deleted from the list. However, it is still possible to miss some relevant literature papers. One such instance is the existence of gray literature such as technical reports, MSc and PhD theses, and company journals. In our case, this literature can be important if the authors report the complete study and validated it by using a case study. In this mapping study, we do not include such information. Another risk of the incompleteness is that the searches on electronic databases are inconsistent in search engines. Those databases have limited capabilities in terms of performing complex search strings. This could lead to irrelevant studies being selected. Therefore, we define a selection criteria and applied inclusion/exclusion procedures on primary studies manually. Thereby, we try to reduce the publication bias and search bias as much as possible by adopting the guidelines and defining criteria.

After the primary studies selected and evaluated, we perform the data extraction in order to derive the review result. In this process, if data extraction is not modeled in a well-defined way, this can cause data extraction bias. In order to define the data extraction model, we read a set of randomly selected papers. Each of them was used to construct initial data extraction form based on previously defined research questions and we perform pilot data extraction on randomly selected primary studies. After the pilot data extraction process, we add some fields to the form in order to capture relevant results. Furthermore, to eliminate the unnecessary or irrelevant results, we remove some fields from the data extraction form. To reduce the data extraction bias, we apply this several times and after a number of iterations and discussions, we constructed the final data extraction model.

Another one of the main threats to the validity of this systematic mapping study is the classification of primary studies. To deal with this bias, the classification process is done based on what has been written and claimed by the corresponding authors instead of judging each paper independent from the authors statements. This has the advantage that the primary studies can also be examined by other reviewers, and this will support the reliability and reproducibility of our SMS.

5 Related work

Rafi et al. (2012) provide a systematic literature review and survey on benefits and limitations of automated software testing. They consider the 25 studies and conduct data extraction for both benefits and limitations of automated software testing. They use the data extraction results to conduct practitioner survey to determine whether the benefits and limitations are of relevance for the software industry at large. They share the survey with 115 people from software industry. With the survey results, they conclude that main benefits of test automation are reusability, repeatability, and effort saved in test executions. Moreover, the participants of the survey indicate that automation of testing improvise the test coverage. On the other hand, for the limitations, they find that designing test cases and buying a test automation tool and training the staff is highly costly. In this work, the authors only focus on the benefits and limitation of automated testing where our work focuses on not only benefits and limitation of the MBT but also the proposed solution approaches to provide more detailed investigation of MBT approaches.

In the study of Hartman et al. (2007), the authors conduct a survey on the different types of languages used to define test models. In this work, they indicate that one of the key factors in model-based testing is language used to define test models. They compare the languages from different aspects. First, design languages and test-specific languages are compared and advantages/disadvantages of these languages are presented. Secondly, they compare the domain-specific languages and generic modeling languages and discuss the issues in each language. In addition, they compare the visual and textual languages for test modeling. Then, they discuss the commercial, in-house, and open-source tools. Finally, they give the pros and cons of the standard languages and custom-made proprietary solutions. In this study, the authors only focus on specific part of model-based testing. They only consider the test model construction step (explained in 2.1) of model-based testing. However, in our work, we examine each of the model-based testing steps and provide more detailed analysis on each of the steps.

In Dias Neto et al. (2007), they perform a systematic review on model-based testing approaches. They focus on 78 papers to show where MBT approaches have been applied, the characteristics and the limitations of MBT. They analyze the selected studies in terms of application scope of MBT approaches, level of automation, tools to support MBT, models used for test case generation, test coverage criteria, behavior model limitations, cost and complexity of application of MBT approach. In addition to these, they also discuss the issues regarding MBT approaches and limitation of MBT approaches. With this work, the authors analyze the all studies in the literature, where we only consider to take the studies which focus on only software safety. Our work also has slightly different fields for data extraction where it is based on the model-based testing process (explained in 2.1).

6 Conclusions

In this paper, we have presented the results of a systematic mapping study on model-based testing (MBT) for software safety. Our main goal was to investigate the application domains in which MBT for safety was applied, identify the current challenges and research directions, and identify the potential solutions in this context. To the best of our knowledge, no previous systematic mapping study has been performed yet on MBT and for this purpose. The SMS is carried out meticulously and included the published literature since 2005. We can identify 604 papers from the searching literature, and 36 of them are found as relevant primary studies to our research questions.

Based on this mapping study, we have analyzed the current MBT approaches for software safety and presented the results to help the researchers and identify the future research directions. From our study, we can conclude that MBT for safety is quite broad and has been applied in various application domains. This shows that MBT for safety is indeed relevant and can help testing safety-critical systems. We could identify recurring common motivations for adopting MBT, such as reducing the cost of testing and increasing test coverage.

All the selected primary studies show that existing MBT approaches have a clear impact on software safety testing. In parallel, we could observe that the proposed solution approaches are focused on particular application domains. The focus on specific application domains is often necessary to better support the targeted generation of test cases in that application domain. On the other hand, the proposed MBT for safety approach as such is not directly applicable to other application domains and as such is less generic and less reusable. Besides the focus on particular application domains, we could also observe that the MBT for software safety approaches usually focus on a small scope of the system. A model of the complete system is often missing and likewise an overall evaluation of the system using MBT only is not feasible. We could thus identify that the current MBT for software safety approaches is basically focused on particular application domains and has a smaller system scope. In this context, we could also observe that only a few studies perform an abstract test case generation step that is useful for defining generic and reusable test cases.

An important aspect of MBT is of course the adoption of tool support to provide the necessary automation. Most of the approaches have indeed tool support for model and test case generation. In alignment with the focus of the solution approaches, the corresponding tools also focus on particular application domains. Further, the learning curve for the tools is different.

To sum up, we have highlighted the relevant primary studies in the state-of-the-art regarding MBT for safety. The SMS provides both an analysis and synthesis of the results and can help to provide a vision for further research. In our future work, we will indeed tackle some of the identified challenges to enhance MBT for safety.

Appendix A Search Strings

Table 11 Search strings

Electronic database	Search string
IEEE Xplore	((“Abstract”: model based software test OR “Abstract”: model driven software test) AND “Abstract”: safety)
ACM Digital Library	((Title:“model based testing” OR Title:“model based software testing” OR Title:“model-based testing” OR Title:“model-based software testing” OR Title:“model driven testing” OR Title:“model driven software testing” OR Title:“model-driven testing” OR Title:“model-driven software testing” OR Title:“model based test” OR Title:“model based software test” OR Title:“model-based test” OR Title:“model-based software test” OR Title:“model driven test” OR Title:“model driven software test” OR Title:“model-driven test” OR Title:“model-driven software test”) AND Title:“safety”) OR ((Abstract:“model based testing” OR Abstract:“model based software testing” OR Abstract:“model-based testing” OR Abstract:“model-based software testing” OR Abstract:“model driven testing” OR Abstract:“model driven software testing” OR Abstract:“model-driven testing” OR Abstract:“model-driven software testing” OR Abstract:“model based test” OR Abstract:“model based software test” OR Abstract:“model-based test” OR Abstract:“model-based software test” OR Abstract:“model driven test” OR Abstract:“model driven software test” OR Abstract:“model-driven test” OR Abstract:“model-driven software test”) AND Abstract:“safety”)
Wiley Interscience	(“model based testing” OR “model based software testing” OR “model-based testing” OR “model-based software testing” OR “model driven testing” OR “model driven software testing” OR “model-driven testing” OR “model-driven software testing” OR “model based test” OR “model based software test” OR “model-based test” OR “model-based software test” OR “model driven test” OR “model driven software test” OR “model-driven test” OR “model-driven software test”) AND “software” AND “safety”
Science Direct	TITLE-ABSTR-KEY ((“model based testing” OR “model based software testing” OR “model-based testing” OR “model-based software testing” OR “model driven testing” OR “model driven software testing” OR “model-driven testing” OR “model-driven software testing” OR “model based test” OR “model based software test” OR “model-based test” OR “model-based software test” OR “model driven test” OR “model driven software test” OR “model-driven test” OR “model-driven software test”) AND “safety”)
Springer	(“model based testing” OR “model based software testing” OR “model-based testing” OR “model-based software testing” OR “model driven testing” OR “model driven software testing” OR “model-driven testing” OR “model-driven software testing” OR “model based test” OR “model based software test” OR “model-based test” OR “model-based software test” OR “model driven test” OR “model driven software test” OR “model-driven test” OR “model-driven software test”) AND “safety”
ISI Web of Knowledge	((TI = “model based testing” OR TI = “model based software testing” OR TI = “model-based testing” OR TI = “model-based software testing” OR TI = “model driven testing” OR TI = “model driven software testing” OR TI = “model-driven testing” OR TI = “model-driven software testing” OR TI = “model based test” OR TI = “model based software test” OR TI = “model-based test” OR TI = “model-based software test” OR TI = “model driven test” OR TI = “model driven software test” OR TI = “model-driven test” OR TI = “model-driven software test”) AND TI = “safety”) OR ((TS = “model based testing” OR TS = “model based software testing” OR TS = “model-based testing” OR TS = “model-based software testing” OR TS = “model driven testing” OR TS = “model driven software testing” OR TS = “model-driven testing” OR TS = “model-driven software testing” OR

Table 11 (continued)

Electronic database	Search string
	TS = "model based test" OR TS = "model based software test" OR TS = "model-based test" OR TS = "model-based software test" OR TS = "model driven test" OR TS = "model driven software test" OR TS = "model-driven test" OR TS = "model-driven software test") AND TS = "safety")
Scopus	TITLE-ABS-KEY (("model based testing" OR "model based software testing" OR "model-based testing" OR "model-based software testing" OR "model driven testing" OR "model driven software testing" OR "model-driven testing" OR "model-driven software testing" OR "model based test" OR "model based software test" OR "model-based test" OR "model-based software test" OR "model driven test" OR "model driven software test" OR "model-driven test" OR "model-driven software test") AND "safety")

APPENDIX-B Data Extraction Form

Table 12 Data extraction form

#	Extraction element	Contents
General Information		
1	ID	Unique id for the study
2	SMS Category	○ Include ○Exclude
3	Title	Full title of the article
4	Date of Extraction	The date it is added into repository
5	Year	The publication year
6	Authors	
7	Repository	ACM, IEEE, ISI Web of Knowledge, Science Direct, Springer, Wiley Interscience
8	Type	○ Journal ○ Article ○ Book Chapter
Study Description		
10	Main theme of the study	
11	Motivation for the study	
12	Existence of safety model	○ Yes ○ No
13	Targeted domain	Automation, Automotive, Medical, Nuclear, Power Consumption, Railway, Robotics
14	Requirement specification language	○ Formal ○ Informal ○ Not specified
15	Model specification language	Automata, DSL, UML, Object-Oriented Action Systems, Product Graph, Transition System
16	Method for generating models from requirements	○ Automatic ○ Manual
17	Test selection criteria	Algorithm, Temporal Logic Expression, Not specified
18	Test case definition language	○ Formal ○ Informal ○ Not specified
19	Type of generated test elements	Test Case, Test Script, Test Data, Test Sequence, Test Oracle, Test Scenario
20	Solution approach for generating test elements	Tool , Model Checking, Not Specified, Graph Theory Algorithm, Multi-Object Checking, Mutation, Model Transformation
21	Contribution Type	○ Framework ○ Tool ○ Method
22	Assessment Approach	○ Case Study ○ Experiment
23	Evidence Type	○ Academic Case ○Industrial Case ○ Academic Experiment ○ Industrial Experiment
24	Findings	
25	Constraints / Limitations	
Evaluation		
26	Personal note	The opinions of the reviewer about the study
27	Additional note	Publication details
28	Quality Assessment	Detailed quality scores

APPENDIX-C Study Quality Assessment Form

Table 13 Study quality assessment form

Study	Quality of reporting			Rigor			Credibility		Relevance		*Quality of reporting*	*Rigor*	*Assessment of credibility*	*Relevance*	*Total*
	Q1	*Q2*	*Q3*	*Q4*	*Q5*	*Q6*	*Q7*	*Q8*	*Q9*	*Q10*					
[1]	1.0	1.0	0.5	1.0	1.0	1.0	0.5	0.5	1.0	1.0	2.5	3.0	1.0	2.0	8.5
[2]	1.0	1.0	0.5	0.5	0.5	0.5	0.0	0.5	1.0	0.5	2.5	1.5	0.5	1.5	6.0
[3]	1.0	1.0	1.0	1.0	1.0	1.0	0.5	1.0	1.0	1.0	3.0	3.0	1.5	2.0	9.5
[4]	1.0	1.0	0.5	0.5	0.5	0.5	0.5	0.5	1.0	0.5	2.5	1.5	1.0	1.5	6.5
[5]	1.0	1.0	0.5	1.0	0.5	1.0	0.0	0.5	1.0	0.5	2.5	2.5	0.5	1.5	7.0
[6]	1.0	1.0	0.5	0.5	1.0	1.0	0.0	0.5	1.0	0.5	2.5	2.5	0.5	1.5	7.0
[7]	1.0	1.0	0.5	0.5	1.0	1.0	0.5	0.5	1.0	0.5	2.5	2.5	1.0	1.5	7.5
[8]	1.0	1.0	0.5	1.0	0.5	1.0	0.0	1.0	1.0	0.5	2.5	2.5	1.0	1.5	7.5
[9]	1.0	1.0	0.5	0.5	0.5	1.0	0.0	0.5	1.0	0.5	2.5	2.0	0.5	1.5	6.5
[10]	1.0	1.0	0.5	1.0	1.0	1.0	0.5	1.0	1.0	1.0	2.5	3.0	1.5	2.0	9.0
[11]	1.0	1.0	0.5	1.0	0.5	1.0	0.5	0.5	1.0	0.5	2.5	2.5	1.0	1.5	7.5
[12]	1.0	1.0	0.5	0.5	1.0	1.0	0.5	1.0	1.0	1.0	2.5	2.5	1.5	2.0	8.5
[13]	1.0	1.0	0.5	1.0	1.0	1.0	0.5	1.0	1.0	1.0	2.5	3.0	1.5	2.0	9.0
[14]	1.0	1.0	0.5	0.5	0.5	0.5	0.0	0.5	0.5	1.0	2.5	1.5	0.5	1.5	6.0
[15]	1.0	1.0	0.5	1.0	1.0	0.5	0.0	0.5	1.0	0.5	2.5	2.5	0.5	1.5	7.0
[16]	1.0	1.0	0.5	0.5	0.5	1.0	0.0	1.0	1.0	1.0	2.5	2.0	1.0	2.0	7.5
[17]	1.0	1.0	0.5	1.0	0.5	1.0	0.0	0.5	1.0	1.0	2.5	2.5	0.5	2.0	7.5
[18]	1.0	1.0	0.5	1.0	1.0	1.0	0.0	1.0	1.0	0.5	2.5	3.0	1.0	1.5	8.0
[19]	1.0	1.0	1.0	1.0	0.5	1.0	0.0	1.0	1.0	1.0	3.0	2.5	1.0	2.0	8.5
[20]	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	3.0	3.0	1.0	2.0	9.0
[21]	1.0	1.0	1.0	1.0	0.5	1.0	0.0	0.5	1.0	0.5	3.0	2.5	0.5	1.5	7.5
[22]	1.0	1.0	1.0	1.0	0.5	1.0	0.0	1.0	1.0	1.0	3.0	2.5	1.0	2.0	8.5
[23]	1.0	1.0	0.5	0.5	0.5	0.5	0.0	0.5	1.0	0.5	2.5	1.5	0.5	1.5	6.0
[24]	1.0	0.5	0.5	0.5	0.5	0.5	0.0	0.5	1.0	0.5	2.0	1.5	0.5	1.5	5.5
[25]	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	3.0	3.0	1.0	2.0	9.0
[26]	1.0	1.0	0.5	1.0	0.5	0.5	0.0	0.5	1.0	0.5	2.5	2.0	0.5	1.5	6.5
[27]	1.0	1.0	0.5	1.0	0.5	1.0	0.0	1.0	1.0	1.0	2.5	2.5	1.0	2.0	8.0
[28]	1.0	1.0	1.0	1.0	1.0	0.5	0.0	0.5	0.5	0.5	3.0	2.5	0.5	1.0	7.0

Table 13 (continued)

Study	Quality of reporting			Rigor			Credibility		Relevance		*Quality of reporting*	*Rigor*	*Assessment of credibility*	*Relevance*	*Total*
	Q1	*Q2*	*Q3*	*Q4*	*Q5*	*Q6*	*Q7*	*Q8*	*Q9*	*Q10*					
[29]	1.0	1.0	1.0	1.0	1.0	0.5	0.0	1.0	0.5	0.5	3.0	2.5	1.0	1.0	7.5
[30]	1.0	1.0	1.0	1.0	0.5	0.5	0.0	1.0	1.0	1.0	3.0	2.0	1.0	2.0	8.0
[31]	1.0	1.0	1.0	1.0	0.5	1.0	0.0	1.0	1.0	1.0	3.0	2.5	1.0	2.0	8.5
[32]	1.0	1.0	1.0	1.0	0.5	1.0	0.0	1.0	0.5	0.5	3.0	2.5	1.0	1.0	7.5
[33]	0.5	0.5	1.0	1.0	0.5	1.0	0.0	0.5	1.0	0.5	2.0	2.5	0.5	1.5	6.5
[34]	1.0	1.0	1.0	0.5	1.0	1.0	0.0	1.0	1.0	0.5	3.0	2.5	1.0	1.5	8.0
[35]	1.0	1.0	0.5	0.5	0.5	1.0	0.0	0.5	1.0	0.5	2.5	2.0	0.5	1.5	6.5
[36]	1.0	1.0	0.5	1.0	1.0	1.0	0.0	1.0	1.0	0.5	2.5	3.0	1.0	1.5	8.0

References

Aichernig, B. K., Brandl, H., Jöbstl, E., & Krenn, W. (2011). UML in action: A two-layered interpretation for testing. *SIGSOFT Softw Eng. Notes, 36*(1), 1–8. doi:10.1145/1921532.1921559.

Aichernig, B. K., Hörmaier, K., & Lorber, F. (2014). Debugging with timed automata mutations. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 8666 LNCS*, 49–64. doi:10.1007/978-3-319-10506-2_4.

Aichernig, B. K., Ničković, D., & Tiran, S. (2015). Scalable incremental test-case generation from large behavior models. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 9154, pp. 1–18). doi:10.1007/978-3-319-21215-9_1.

Arcaini, P., Gargantini, A., & Riccobene, E. (2017). Rigorous development process of a safety-critical system: From ASM models to Java code. *International Journal on Software Tools for Technology Transfer, 19*(2), 247–269. doi:10.1007/s10009-015-0394-x.

ATGT Tool. (n.d.). http://cs.unibg.it/gargantini/software/atgt/. Accessed 3 September 2016.

Atkins, D., Best, D., Briss, P. A., Eccles, M., Falck-Ytter, Y., Flottorp, S., et al. (2004). Grading quality of evidence and strength of recommendations. *BMJ (Clinical Research ed.), 328*(7454), 1490. doi:10.1136/bmj.328.7454.1490.

Auguston, M., Michael, J. B., & Shing, M.-T. (2006). Environment behavior models for automation of testing and assessment of system safety. *Information and Software Technology, 48*(10), 971–980. doi:10.1016/j.infsof.2006.03.005.

Bourque, P., & Dupuis, R. (2004). *Guide to the Software Engineering Body of Knowledge 2004 Version. SWEBOK 2004 Guide to the Software Engineering Body of Knowledge* (Vol. 1). doi:10.1109/SESS.1999.767664.

Choi, Y., & Byun, T. (2017). Constraint-based test generation for automotive operating systems. *Software and Systems Modeling, 16*(1), 7–24. doi:10.1007/s10270-014-0449-6.

Cichos, H., Oster, S., Lochau, M., & Schuerr, A. (2011). Model-based coverage-driven test suite generation for software product lines. *Model Driven Engineering Languages and Systems, 6981*, 425–439.

Dias Neto, A. C., Subramanyan, R., Vieira, M., & Travassos, G. H. (2007). A survey on model-based testing approaches: A systematic review. In *Proceedings of the 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies: Held in Conjunction with the 22Nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007* (pp. 31–36). New York, NY, USA: ACM. doi:10.1145/1353673.1353681.

Enoiu, E. P., Sundmark, D., & Pettersson, P. (2013). Model-based test suite generation for function block diagrams using the UPPAAL model checker. Proceedings–IEEE 6th International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2013, 158–167. doi:10.1109/ICSTW.2013.27.

Enoiu, E. P. ., Čaušević, A. ., Ostrand, T. J. ., Weyuker, E. J. ., Sundmark, D., Pettersson, P. (2014). Automated test generation using model checking: an industrial evaluation. *International Journal on Software Tools for Technology Transfer*, 335–353. doi:10.1007/s10009-014-0355-9.

Fang, L., Kitamura, T., Do, T. B. N., & Ohsaki, H. (2012). Formal model-based test for AUTOSAR multicore RTOS. *Proceedings–IEEE 5th International Conference on Software Testing, Verification and Validation, ICST 2012*, 251–259. doi:10.1109/ICST.2012.105.

Fraser, G., Wotawa, F., & Ammann, P. E. (2009). Testing with model checkers: A survey. *Software Testing Verification and Reliability.* doi:10.1002/stvr.402.

Gargantini, A. (2007). Using model checking to generate fault detecting tests. *Tests and Proofs*, 189–206. doi:10.1007/978-3-540-73770-4_11.

Gario, A., Andrews, A., & Hagerman, S. (2015). Fail-safe testing of safety-critical systems: A case study and efficiency analysis. *Software Quality Journal.* doi:10.1007/s11219-015-9283-5.

Gentile, U., Marrone, S., Mele, G., Nardone, R., & Peron, A. (2014). Test specification patterns for automatic generation of test sequences. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 8718 LNCS*, 170–184. doi:10.1007/978-3-319-10702-8_12.

Grasso, D., Fantechi, A., Ferrari, A., Becheri, C., & Bacherini, S. (2010). Model based testing and abstract interpretation in the railway signaling context. *ICST 2010–3rd International Conference on Software Testing, Verification and Validation*, 103–106. doi:10.1109/ICST.2010.44.

Hartman, A., Katara, M., & Olvovsky, S. (2007). Choosing a test modeling language: A survey. *Hardware and Software, Verification and Testing*, 204–218. doi:10.1007/978-3-540-70889-6_16.

Herber, P., & Glesner, S. (2015). Formal modeling and verification of cyber-physical systems. doi:10.1007/978-3-658-09994-7.

Herzner, W., Schlick, R., Schütz, W., Brandl, H., & Krenn, W. (2010). Towards generation of efficient test cases from UML/OCL models for complex safety-critical systems. *Elektrotechnik und Informationstechnik, 127*(6), 181–186. doi:10.1007/s00502-010-0741-2.

Hessel, A., & Pettersson, P. (2007a). COVER—A real-time test case generation tool.

Hessel, A., & Pettersson, P. (2007b). A global algorithm for model-based test suite generation. *Electronic Notes in Theoretical Computer Science*, *190*(2 SPEC. ISS.), 47–59. doi:10.1016/j.entcs.2007.08.005.

Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2001). Introduction to automata theory, languages, and computation, 2nd edition. *ACM SIGACT News, 32*(1), 60. doi:10.1145/568438.568455.

Jensen, K. (1987). Coloured Petri nets. In Petri nets: central models and their properties (Vol. 254, pp. 248–299). doi:10.1007/BFb0046842.

Kandl, S., Kirner, R., & Puschner, P. (2006). Development of a framework for automated systematic testing of safety-critical embedded systems. *Proceedings of the Fourth Workshop on Intelligent Solutions in Embedded Systems, WISES 2006*, 65–77. doi:10.1109/WISES.2006.237154.

Kim, J. H., Larsen, K. G., Nielsen, B., Mikučionis, M., & Olsen, P. (2015). Formal analysis and testing of real-time automotive systems using UPPAAL tools. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 9128, pp. 47–61). doi:10.1007/978-3-319-19458-5_4.

Kitchenham, B., & Charters, S. (2007). *Guidelines for Performing Systematic Literature Reviews in Software Engineering. EBSE Technical Report Nr. EBSE-2007-01. EBSE Technical Report*. http://www.dur.ac.uk/ebse/resources/Systematic-reviews-5-8.pdf

Kloos, J., & Eschbach, R. (2010). A systematic approach to construct compositional behaviour models for network-structured safety-critical systems. *Electronic Notes in Theoretical Computer Science, 263*, 145–160. doi:10.1016/j.entcs.2010.05.009.

Kloos, J., Hussain, T., & Eschbach, R. (2011). Risk-based testing of safety-critical embedded systems driven by fault tree analysis. *Proceedings–4th IEEE International Conference on Software Testing, Verification, and Validation Workshops, ICSTW 2011*, 26–33. doi:10.1109/ICSTW.2011.90.

Kollmann, M., & Hon, Y. M. (2007). Generating scenarios by multi-object checking. *Electronic Notes in Theoretical Computer Science*, *190*(2 SPEC. ISS.), 61–72. doi:10.1016/j.entcs.2007.08.006.

Krenn, W., Schlick, R., & Aichernig, B. K. (2010). Mapping UML to labeled transition systems for test-case generation: A translation via object-oriented action systems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *6286 LNCS*, 186–207. doi:10.1007/978-3-642-17071-3_10.

Leskovec, J., Rajaraman, A., & Ullman, J. D. (2014). Mining social-network graphs. *Mining of Massive Datasets*, 340–393. doi:10.1017/CBO9781139924801.011.

Li, M., & Kumar, R. (2011). Stateflow to extended finite automata translation. In *Proceedings–International Computer Software and Applications Conference* (pp. 1–6). doi:10.1109/COMPSACW.2011.11.

Lochau, M., & Goltz, U. (2010). Feature interaction aware test case generation for embedded control systems. *Electronic Notes in Theoretical Computer Science, 264*(3), 37–52. doi:10.1016/j.entcs.2010.12.013.

Lv, J., Li, K., Wei, G., Tang, T., Li, C., & Zhao, W. (2013). Model-based test cases generation for onboard system. *2013 I.E. Eleventh International Symposium on Autonomous Decentralized Systems (ISADS)*, 1–6. doi:10.1109/ISADS.2013.6513433.

Marrone, S., Flammini, F., Mazzocca, N., Nardone, R., & Vittorini, V. (2014). Towards model-driven V&V assessment of railway control systems. *International Journal on Software Tools for Technology Transfer, 16*(6), 669–683. doi:10.1007/s10009-014-0320-7.

MaTeLo Tool. (n.d.). http://www.all4tec.net/Matelo/model-based-testing.html. Accessed 10 March 2017.

Mathaikutty, D. A., Ahuja, S., Dingankar, A., & Shukla, S. (2007). *Model-driven test generation for system level validation* (pp. 83–90). HLDVT: Proceedings–IEEE International High-Level Design Validation and Test Workshop. doi:10.1109/HLDVT.2007.4392792.

Micskei, Z., Szatmári, Z., Oláh, J., & Majzik, I. (2012). A concept for testing robustness and safety of the context-aware behaviour of autonomous systems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *7327 LNAI*, 504–513. doi:10.1007/978-3-642-30947-2_55.

Mohalik, S., Gadkari, A., Yeolekar, A., Shashidhar, K. C., & Ramesh, S. (2014). Automatic test case generation from Simulink/Stateflow models using model checking. *Software Testing Verification and Reliability, 24*(2), 155–180. doi:10.1002/stvr.1489.

NuSMV Language. (n.d.). http://nusmv.fbk.eu/. Accessed 20 January 2014.

Petersen, K., Vakkalanka, S., & Kuzniarz, L. (2015). Guidelines for conducting systematic mapping studies in software engineering: An update. In *Information and Software Technology* (Vol. 64, pp. 1–18). doi:10.1016/j.infsof.2015.03.007.

Proetzsch, M., Zimmermann, F., Eschbach, R., Kloos, J., & Berns, K. (2010). A systematic testing approach for autonomous mobile robots using domain-specific languages. *KI 2010: Advances in Artificial Intelligence, 6359*, 317–324. doi:10.1007/978-3-642-16111-7_36.

Prowell, S. J. (2003). JUMBL: A tool for model-based statistical testing. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences, HICSS 2003*. doi:10.1109/HICSS.2003.1174916.

Rafi, D. M., Moses, K. R. K., Petersen, K., & Mäntylä, M. V. (2012). Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. In *2012 7th International Workshop on Automation of Software Test, AST 2012–Proceedings*. doi:10.1109/IWAST.2012.6228988.

Samih, H., Guen, H. Le, Bogusch, R., Acher, M., & Baudry, B. (2014). Deriving usage model variants for model-based testing: An industrial case study. *Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems, ICECCS*, 77–80. doi:10.1109/ICECCS.2014.19.

Schrammel, P., Melham, T., & Kroening, D. (2016). Generating test case chains for reactive systems. *International Journal on Software Tools for Technology Transfer, 18*(3), 319–334. doi:10.1007/s10009-014-0358-6.

Spin–formal verification. (n.d.). http://spinroot.com/spin/whatispin.htm. Accessed 21 January 2014.

Stallbaum, H., & Rzepka, M. (2011). Toward DO-178B-compliant test models. *Proceedings–2010 Workshop on Model-Driven Engineering, Verification, and Validation, MoDeVVa 2010*, 25–30. doi:10.1109/MoDeVVa.2010.21.

Symbolic Analysis Laboratory Title. (n.d.). http://sal.csl.sri.com/. Accessed 21 January 2014.

Thomas, F., Delatour, J., Terrier, F., & Gérard, S. (2008). Toward a framework for explicit platform-based transformations. In *Proceedings–11th IEEE Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, ISORC 2008* (pp. 211–218). doi:10.1109/ISORC.2008.64.

Tseng, W.-H., & Fan, C.-F. (2013). Systematic scenario test case generation for nuclear safety systems. *Information and Software Technology, 55*(2), 344–356. doi:10.1016/j.infsof.2012.08.016.

Utting, M., Legeard, B., Pretschner, A., & Legeard, B. (2006). A taxonomy of model-based testing. *Software Testing, Verification and Reliability, 22*(April), 297–312. doi:10.1002/stvr.456.

Wilkinson, T., Butler, M., & Colley, J. (2014). A systematic approach to requirements driven test generation for safety critical systems, 43–56. doi:10.1007/978-3-319-12214-4_4.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). Experimentation in software engineering. *Experimentation in Software Engineering, 9783642290*. doi:10.1007/978-3-642-29044-2.

Yu, G., & Xu, Z. W. (2010). Model-based safety test automation of safety-critical software. *2010 International Conference on Computational Intelligence and Software Engineering, CiSE 2010*, (60674004), 4–6. doi:10.1109/CISE.2010.5676883.

Yu, G., Xu, Z. W., & Du, J. W. (2009). An approach for automated safety testing of safety-critical software system based on safety requirements. *Proceedings–2009 International Forum on Information Technology and Applications, IFITA 2009, 3*, 166–169. doi:10.1109/IFITA.2009.18.

Zhang, H., Babar, M. A., & Tell, P. (2011). Identifying relevant studies in software engineering. *Information and Software Technology, 53*(6), 625–637. doi:10.1016/j.infsof.2010.12.010.

Zheng, W., Liang, C., Wang, R., & Kong, W. (2014). Automated test approach based on all paths covered optimal algorithm and sequence priority selected algorithm. *IEEE Transactions on Intelligent Transportation Systems, 15*(6), 2551–2560. doi:10.1109/TITS.2014.2320552.

Havva Gulay Gurbuz Havva received her B.S. degree (2012) in Computer Engineering from Hacettepe University, Turkey and her MSc degree (2014) from Bilkent University, Turkey. Currently she is a software engineer at Microsoft, USA, and PhD student at Wageningen University, The Netherlands.

Bedir Tekinerdogan Prof. Tekinerdogan received his MSc degree (1994) and a PhD degree (2000) in Computer Science, both from the University of Twente, The Netherlands. From 2003 until 2008 he was a faculty member at University of Twente, after which he joined Bilkent University until 2015. At Bilkent he has founded and led the Bilkent Software Engineering Group which aimed to foster research and education on software engineering in Turkey. Currently he is full professor and chair of the Information Technology group at Wageningen University, The Netherlands.