Information and Software Technology 102 (2018) 30–48

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Model-driven architecture based testing: A systematic literature review

Burak Uzun, Bedir Tekinerdogan*

Information Technology Group, Wageningen University, Wageningen, The Netherlands

ARTICLE INFO

Keywords:
Model-based testing
Software architecture
Systematic review

ABSTRACT

Context: Model-driven architecture based testing (MDABT) adopts architectural models of a system under test and/or its environment to derive test artifacts. In the literature, different MDABT approaches have been provided together with the corresponding lessons results and lessons learned.
Objective: The overall objective of this paper is to identify the published concerns for applying MDABT, identify the proposed solutions, and describe the current research directions for MDABT.
Method: To this end we have provided a systematic literature review (SLR) that is conducted by a multi-phase study selection process using the published literature in major software engineering journals and conference proceedings.
Results: We reviewed 739 papers that are discovered using a well-planned review protocol, and 31 of them were assessed as primary studies related to our research questions. Based on the analysis of the data extraction process, we discuss the primary trends and approaches and present the identified obstacles.
Conclusion: This study shows that although a generic process the approaches different in various ways with different goals, modeling abstractions and results. Further, based on the synthesis process in the SLR we can state that the potential of MDABT has not been fully exploited yet.

1. Introduction

Software testing is a process of investigating a software product to identify possible mismatches between expected and present requirements of the system [1,15]. One of the main motivations of software testing is to ensure the correctness of a software system. Software is correct if and only if each valid input to the system produces an output according to system specifications. Therefore, software must be verified and validated according to the provided specifications. Moreover, software testing requires executions of test cases which can detect possible bugs, errors and defects.

In general, exhaustive testing is not practical or tractable for most real programs due to the large number of possible inputs and sequences of operations [21]. As a result, selecting the set of test cases which can detect possible flaws of the system is the key challenge in software testing [8,22].

Model based testing (MBT) addresses this challenge by automating the generation and execution of test cases using models based on system requirements and behavior [14,21,28]. A number of benefits of MBT have been identified in the literature [22]. The main benefits of MBT include improved test coverage, reduced testing time, increased reliability, reusability of tests, increased confidence in the system, less human effort, reduction in cost, increased fault detection, and improved product quality. Altogether MBT is now considered a promising and effective approach for software testing.

MBT relies on models to automate the generation of the test cases and their execution [10]. A model is usually an abstract, partial presentation of the desired behavior of a system under test (SUT). MBT can use different representations of the system to generate testing procedures for different aspects of the software systems. Example models include finite state machines (FSMs), Petri Nets, I/O automata, and Markov Chains [10]. A recent trend in MBT is to adopt software architecture models to provide automated support for the test process [16] leading to the notion of model-driven architecture-based testing (MDABT). Software architecture is different from the other design representations since it provides a gross-level representation of the system at the higher abstraction level [4,25].

So far, various MDABT approaches have been introduced but no explicit effort has been undertaken to provide a systematic overview on the existing literature. Hence, the effectiveness of MDABT, the common approaches as a well as the variations, and the guidelines and lessons learned are actually scattered over different studies in the literature. In this paper we aim to address the following research questions:

(1) What are the addressed concerns for applying model-driven software architecture based testing?

* Corresponding author.
E-mail addresses: burak.uzun@wur.nl (B. Uzun), bedir.tekinerdogan@wur.nl (B. Tekinerdogan).

https://doi.org/10.1016/j.infsof.2018.05.004
Received 30 July 2017; Received in revised form 7 May 2018; Accepted 12 May 2018
Available online 18 May 2018
0950-5849/

(2) What are the proposed solutions in architecture-based testing?
(3) What are the existing research directions within architecture-based testing?

To provide answers to these questions we have adopted a systematic literature review (SLR) approach based on Kitchenham's guidelines [12,13]. We reviewed 739 papers that are discovered using a well-planned review protocol, and 31 of them were assessed as primary studies related to our research questions. Based on the analysis of the data extraction process, we discuss the primary trends and approaches and present the identified obstacles.

For researchers, this SLR gives an overview of the reported MDABT together with an understanding of the common process and the lessons learned and guidelines for future studies. Practitioners may benefit from the SLR 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 of the paper describes the overall background on architecture-based testing, architecture modeling and systematic literature reviews. System 3 discusses the overall protocol of the adopted in SLR. Section 4 presents the results of the adopted SLR protocols. Section 5 presents the related work. Finally, Section 6 presents the conclusion of this study.

2. Background

2.1. Model-based testing

Historically, models have had a long tradition in software engineering and have been widely used in software projects. The primary reason for modeling is usually defined as a means for communication, analysis or guiding the production process. Models are different in nature and quality. Mellor et al. make a distinction between three kinds of models, depending on their level of precision. A model can be considered as a sketch, as a blueprint, or as an executable. A sketch has the level of precision and is typically an informal diagram that is used for creating ideas in the design process. A blueprint is a design with sufficient detail that can be handed over to a developer for realizing the system according to the blueprint. Unlike a sketch and a blueprint, an executable model has everything required to produce the desired functionality of a single domain and can be interpreted by model compilers. In model-driven software development the concept of models can be considered as executable models as defined by the above characterization of Mellor et al. [7,17]. This is in contrast to model-based software development in which models are used as blueprints at the most. The language in which models are expressed is defined by meta-models. As such, a model is said to be an instance of a meta-model, or a model conforms to a meta-model. A meta-model itself is a model that conforms to a meta-meta-model, the language for defining meta-models.

Model-based testing builds on model-driven development in which models are used to automate the testing process. According to Utting et al. [27,28] MBT is a type of testing that utilizes the information in model which is the intended behavior of the system and its environment. There are several motivations for performing model based testing such as easy test maintenance, automated test design and enhancing test quality. In Fig. 1 (adapted from [27]) the process of model based testing is presented. Model of the system under test is constructed from the requirements of the system. Likewise, test selection criteria are formed by requirements, which is used for selecting test cases that detects faults, errors and possibly failures. Test case specifications are constructed from test selection criteria which are then used with system model to generate actual test cases. Test cases are executed at system under test and test results are analyzed by test verdict.

2.2. Software architecture modeling

Software architecture is defined as the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution [11]. Every system has a set of interested stakeholders with their specific concerns. Concerns can be functional or non-functional [24,18]. These stakeholders and their concerns are considered as the architectural drivers that shape the architecture [4,25]. A stakeholder is defined as an individual, team, or organization with interests in, or concerns relative to, a system. Each of the stakeholders' concerns impacts the early design decisions that the architect makes. A common practice is to model different architectural views for describing the architecture according to the stakeholders' concerns [5]. An architectural view is a representation of a set of system elements and relations associated with them to support a concern. Having multiple views helps to separate the concerns and as such support the modeling, understanding, communication and analysis of the software architecture for different stakeholders.

Architectural views conform to viewpoints that represent the conventions for constructing and using a view. An example viewpoint that is often used is the decomposition viewpoint that defines the specific guidelines and notations for representing the overall decomposition of the system. Another example is the deployment viewpoint that defines its own guidelines and modeling notations for mapping software modules and components to non-software elements such as hardware nodes. In the literature, initially a fixed set of viewpoints have been proposed to document the architecture. Because of the different concerns that need to be addressed for different systems, the current trend recognizes that the set of views should not be fixed but multiple viewpoints might be introduced instead. The ISO/IEC 42010 standard Recommended Practice for Architectural Description [11] indicates in an abstract sense that an architecture description consists of a set of views, each of which conforms to a viewpoint realizing the various concerns of the stakeholders. The Views and Beyond (V&B) approach as proposed by Clements et al. is another multi-view approach [4] that proposes the notion of architectural style similar to the notion of architectural viewpoint.

2.3. Systematic reviews

The overall objective of this paper is to identify, analyze and describe the state of the art advances in MDABT. For this we will apply an SLR which is a well-defined and rigorous method to identify, evaluate and interpret all relevant studies regarding a particular research question, topic area or phenomenon of interest [12,13]. The goal of an SLR is to give a fair, credible and unbiased evaluation of a research topic using a trustworthy, rigorous and auditable method. There are several reasons for undertaking a systematic literature review including summarizing the existing evidence concerning a treatment or technology, identifying any gaps in current research in order to suggest areas for further investigation, providing a framework/background in order to appropriately position new research activities, examining the extent to which empirical evidence supports/contradicts theoretical hypotheses, or assisting in the generation of new hypotheses. Different approaches have been presented in the literature for conducting SLRs in different domains. We followed the complete guidelines for performing SLRs as proposed by Kitchenham et al. [13]. In the following subsections we discuss the applied research method that is based on an extensive review protocol.

3. Research method

In this section, we describe the SLR research methodology that we have adopted. First, we describe the adopted review protocol in Section 3.1 and then proceed with the execution of the steps of the

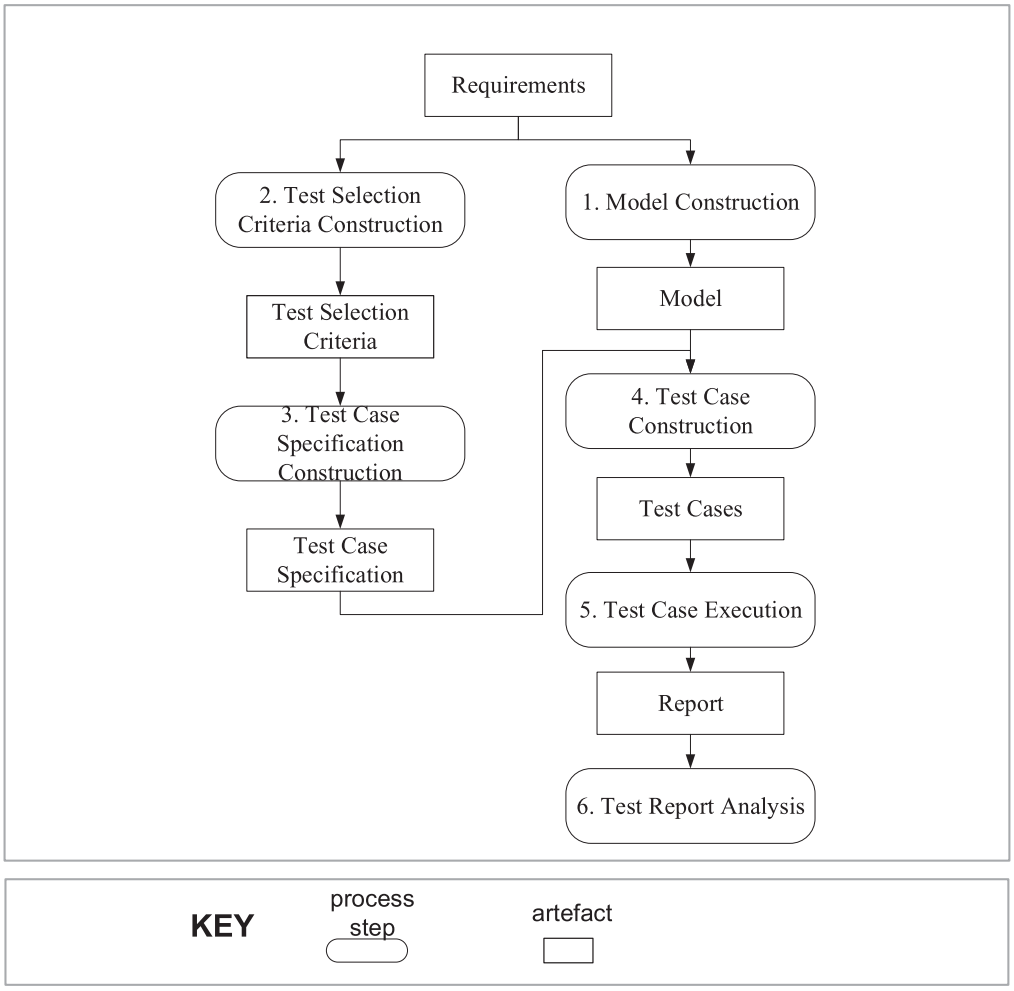


Fig. 1. Process of model based testing.

protocol in the subsequent sub-sections.

3.1. Review protocol

The adopted review protocol based on [13] is shown in Fig. 2. According to the guidelines it includes the phases including planning the review, conducting the review and documenting the review.

Planning the review results in the definition of the protocol. In essence we have defined this by studying the guidelines of Kitchenham et al. The conducting of review consists of five separate steps. First, we specified our research questions based on the objectives of this systematic review. Based on this we followed the selection of the studies. For this we defined the search scope and the search strategy. The search scope defines the time span and the venues that we looked at. In the search strategy, we devised the search strings that were formed after performing deductive pilot searches. A good search string brings the

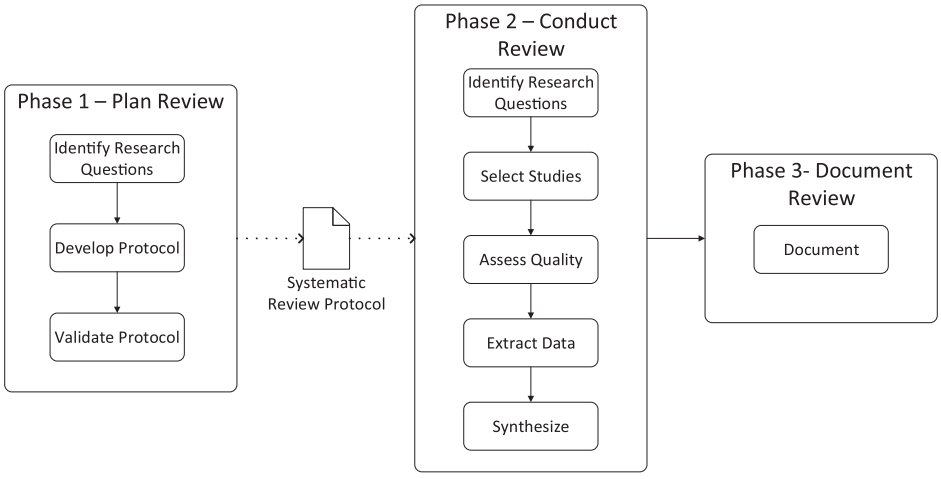


Fig. 2. The adopted review protocol.

appropriate search results that will come to a successful conclusion in terms of sensitivity and precision rates. Once the search strategy was defined, we specified the study selection criteria that are used to determine which studies are included in, or excluded from, the systematic review. The selection criteria were piloted on a number of primary studies. We screened the primary studies at all phases on the basis of inclusion and exclusion criteria. Also, peer reviews were performed by the authors throughout the study selection process. The process followed with quality assessment in which the primary studies that resulted from the search process were screened based on quality assessment checklists and procedures. Once the final set of preliminary studies were defined the data extraction strategy was developed. For this we developed a data extraction form that was defined after a pilot study. In the final step the data synthesis process took place in which we have presented the extracted data and associated results. The review results were documented in the last phase, *Document Review*, of the overall process.

3.2. Research questions

The fundamental step of the SLR is the identification of specific and valid research questions. The research questions drive the entire review process providing the basis for both the decision of the selection of primary studies, and the decision on what data must be extracted and how the data is synthesized to answer the questions. We have identified the following research questions:

RQ1: What are the addressed concerns for applying model-driven software architecture based testing?
RQ2: What are the proposed solutions in model-driven architecture-based testing?
RQ3: What are the existing research directions within model-driven architecture-based testing?

3.3. Search strategy

The search process begins by first defining a search strategy. For this the basic scope of the search strategy must be determined, which include the specific sources that will be searched and the search strings that will be used for automated searches and the sources that will be searched manually. Our search scope includes two attributes which are publication date range and publication platforms. We have selected studies between 2000 and April 2017 for our publication date range attribute. The start date of 2000 has been selected because the first paper having strong foundations on the topic was published at this date. For the publication platforms we included IEEE Xplore, ACM Digital Library, Wiley, Science Direct, Springer, and ISI Web of Knowledge. To search a database, we used both automated search and manual search. Automated search is performed by executing search strings on search engines of electronic data sources. Manual search is conducted by manually browsing journals, conference proceedings or other important sources.

We have identified a search string for each publication platform listed in our search scope for retrieving the relevant studies. Each platform has different features, attributes to query for primary studies we are interested in. Hence, we have defined queries for each platform using each the platform search language. The created queries define the intersection of the papers that has software architecture testing and model based testing in the publications title or abstract. The search strings for each platform are described in Appendix A. Table 1 presents the results of the overall search process. The first column provides the searched venues. In the second column of the table, 739 studies are retrieved after executing the search strings in the platforms. The third and fourth columns show the filtered studies after applying the study selection criteria as explained in the following sub-section. In the last step of the process 31 studies have been identified as primary studies to be used for detailed data analysis and synthesis.

Table 1
Overview of search results and study selection.

Source	Number of included studies after applying search query	Number of included studies after EC1-EC4 applied	Number of included studies after EC5-EC8 applied
IEEE Xplore	64	47	12
ACM Digital Library	155	13	5
Wiley Interscience	67	0	0
Science Direct	57	15	5
Springer	155	10	6
ISI Web of Knowledge	76	35	3
Total	739	120	31

3.4. Study selection criteria

The search query strings have been defined so that we will not miss any related studies. On the other hand, study selection criteria have been defined so that irrelevant studies are excluded. We have applied the following exclusions criteria:

EC 1: Papers in which the full text is unavailable.
EC 2: Papers gathered as duplicate or similar at different platforms.
EC 3: Papers are not written in English.
EC 4: Papers do not relate to architecture based testing.
EC 5: Papers do not explicitly discuss architecture based testing.
EC 6: Papers which are experience and survey papers.
EC 7: Papers do not provide a process model for architecture based testing.

3.5. Study quality assessment

Once we have identified the primary studies we have also assessed the quality of each study. For this we had to decide on the criteria against which quality will be assessed. Further we had to establish the procedure for applying the criteria. The criteria have been expressed as a checklist as shown in Table 2.

The procedure for applying the quality criteria is specified in a way that aims, as far as is possible, to ensure the reliability of the assessment. We have applied the following procedure:

1. For each paper, a reviewer was nominated randomly as data extractor/quality assessor or data checker.
2. The data extractor/quality assessor read the paper and completed a form.

Table 2
Adopted quality checklist.

No	Question
Q1	Are the aims of the study 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	Are the variables used in the study likely to be valid and reliable?
Q5	Is the research process documented adequately?
Q6	Are all the study questions answered?
Q7	Are the negative findings presented?
Q8	Are the main findings stated clearly in terms of creditability, validity and reliability?
Q9	Do the conclusions relate to the aim of the purpose of study?
Q10	Does the report have implications in practice and results in research area for model-driven software architecture testing?

3. The checker read the paper and checked the form.
4. Disagreements were resolved by discussion among the researchers.

We utilized a three-point scale including yes (1), somewhat (0.5) and no (0). The results for each primary study filtered by study selection criteria is presented in Appendix-B.

3.6. Data extraction

Data extraction is performed by reading all the 31 selected primary studies for answering each research question. Furthermore, a data extraction form is designed for retrieving all the information to answer the research questions and all the attributes for study quality assessment criteria. The data extraction form contains the set of attributes such as identification number of the study, date of data extraction year, publication year, authors of the study, platform of the publication, and type of the publication. The extraction of data purpose columns is inserted in to the form as well by study description and evaluation parts which can be seen in Appendix-D.

3.7. Data synthesis

Data synthesis is the most important part of the SLR process in which the extracted data from the primary studies is summarized and research questions are answered. In this study, we implemented both qualitative and quantitative synthesis on the extracted. We examined if the qualitative results enable us to clarify any quantitative results as well. The results of the synthesis are provided in the next section.

4. Results

4.1. Overview of the reviewed studies

This section of the study presents the publication year distribution and the publication platforms of the 31 selected primary studies. Fig. 3 shows the publication year distribution of the selected primary studies.

Table 3 presents the publication sources and channels of the selected studies together with the publication type and distribution of studies over the attributes. From the table we can infer that selected primary studies are published in various reputable publication sources such as IEEE, ScienceDirect, ACM and Springer.

4.2. Research methods

In essence, every primary study will have its own research method together with the adopted empirical validations. Table 4 presents the adopted types of the research method that is applied in the selected primary studies.

Three types of research methods are identified during the review process which are case study, experiment and small example. From the table, we can conclude that case study research method is the most frequently adopted approach in the selected studies. On the other hand, one of the studies does not validate the proposed approach via any research method.

4.3. Methodological quality

This section provides the quality of the selected primary studies based on the attributes of relevance, quality of reporting, rigor and credibility [13]. The values for the quality attributes are derived from the quality checklist which was given in the previous section. The first three questions of the quality checklist relate to the reporting quality attribute, the fourth, fifth and sixth questions are used for the calculation of rigor quality, the seventh and eight questions are the assessment questions for credibility quality, and finally the last two questions are used for relevance quality. In Appendix-C the result of the quality checklist is presented. Table 5 shows the qualities for studies in each category.

Table 5a shows the reporting quality of the studies according to the first three questions of the quality checklist. It can be seen that almost all of the primary studies have the highest score while one of the primary studies is close to the highest score. Rigor quality of the study refers to the trustiness of findings of the study. Table 5b shows that 61% of the studies have the highest score in terms of rigor quality. Moreover, 32% of the studies have been assessed as very good. However, 6% of the studies ranked as good rigor quality. Another quality measure is the relevance quality of the primary studies.

Table 5c shows relevance quality scores calculated from ninth and tenth question of the quality checklist. It is observed that, 39% of the studies is directly and %42 of the studies is mostly relevant to MDABT, where %19 of the studies half relevant. The credibility quality of the studies is calculated by using the seventh and eight questions. Table 5d presents the credibility quality score and distribution of the studies. It can be seen that 87% of the studies calculated as 1 point. Remaining 13% of the studies calculated as 0.5 point. According to our evaluation there is no primary study that has full credibility in terms of evidence. All studies are missing the statement of counter example.

The summary of the overall methodological quality scores of selected primary studies is given in Table 5e. Total quality is calculated by adding up all the quality attributes, which are reporting, relevance, rigor and credibility. The values for the quality attributes are derived from the quality checklist as shown in Table 2 (including 10 questions). For example, for the reporting quality attribute three questions will be used for each of which a value of 0, 0.5 or 1 can be given. Hence in total, reporting quality can have a value of 3.0 as a maximum. The similar procedure has been applied to the other 3 quality attributes. The X-scale of the figures in Table 5 represent the frequency for the indicated values. For example, for the reporting quality 1 study got in total 2.5 points while the other 30 studies got a value of 3. For the overall quality we consider 8.5–9 as high quality, 7.5–8 very good quality and 7 as good quality. It can be seen that 55% studies have high overall quality. Further, 35% of the studies have very good overall quality and 10% of the studies have good overall quality.

4.4. Systems investigated

This section provides the results that are extracted from selected primary studies for answering the research questions specified in the previous sections.

RQ1: What are the addressed concerns for applying model-driven software architecture based testing?

From the analysis of the primary studies we could derive that in

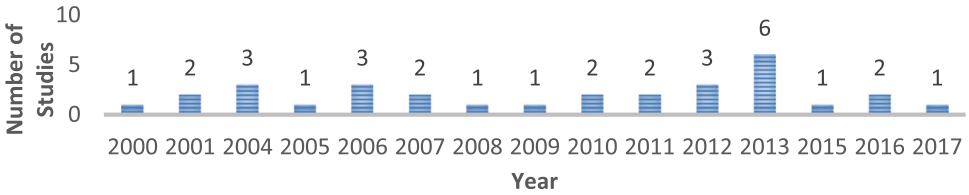


Fig. 3. Year-wise distribution of primary studies.

Table 3
Distribution of the studies over publication channel.

Publication Channel	Publication source	Type	Number of studies
Software Engineering, 2000. Proceedings of the 2000 International Conference	ACM	Article	1
Software Reliability Engineering, 2001. ISSRE 2001. Proceedings. 12th International Symposium	IEEE	Conference	1
Software Engineering, IEEE Transactions on (Volume:30, Issue: 3)	IEEE	Article	1
Fundamental Approaches to Software Engineering	Springer	Article	1
Applying Formal Methods: Testing, Performance, and M/E-Commerce	Springer	Conference	1
Electronic Notes in Theoretical Computer Science	ScienceDirect	Article	1
Proceeding ROSATEA '06 Proceedings of the ISSTA 2006 workshop on Role of software architecture for testing and analysis	ACM	Workshop	1
JSS, Special Edition on Architecting Dependable Systems	ScienceDirect	Article	1
Software Architecture	WebOfKnowledge	Chapter	1
2016 IEEE International Conference on Software Testing, Verification and Validation	IEEE	Conference	1
The 7th International Conference on Ambient Systems, Networks and Technologies	ScienceDirect	Conference	1
Information Technology: New Generations (ITNG), 2010 Seventh International Conference	IEEE	Conference	1
Information and Software Technology			
Volume 55, Issue 7	ScienceDirect	Chapter	1
Journal of Systems and Software Volume 91	ScienceDirect	Article	1
ICSE '01 Proceedings of the 23rd International Conference on Software Engineering	ACM	Conference	1
AST '07 Proceedings of the Second International Workshop on Automation of Software Test	ACM	Workshop	1
ISEC '08 Proceedings of the 1st India software engineering conference	ACM	Conference	1
Software Technologies (ICSOFT), 2015 10th International Joint Conference	IEEE	Conference	1
2010 Forum on Specification & Design Languages (FDL 2010)	IEEE	Conference	1
Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference	IEEE	Conference	1
Applied Machine Intelligence and Informatics (SAMI), 2013 IEEE 11th International Symposium	IEEE	Conference	1
Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference	IEEE	Conference	1
Software Reliability Engineering Workshops (ISSREW), 2012 IEEE 23rd International Symposium	IEEE	Conference	1
Service Science and Innovation (ICSSI), 2013 Fifth International Conference	IEEE	Conference	1
Engineering of Computer-Based Systems, 2007. ECBS '07. 14th Annual IEEE International Conference and Workshops	IEEE	Conference	1
Lecture Notes in Computer Science	Springer	Chapter	1
Communications in Computer and Information Science	Springer	Chapter	1
Software Testing, Verification and Validation Workshops (ICSTW)	WebOfKnowledge	Workshop	1
Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference	WebOfKnowledge	Conference	1
Lecture Notes in Computer Science	Springer	Chapter	1
IFIP Advances in Information and Communication Technology	Springer	Conference	1

Table 4
Distribution of studies over research method.

Research method	Studies	Number	Percent
Case study	A, C, D, E, F, H, M, K, O, R, W, Y, DD, EE	14	45
Experiment	B, L, U	3	10
Small example	G, J, N, P, Q, S, T, V, X, Z, AA, BB, CC	13	42
None	I	1	3

essence two basic concerns are addressed in the application of model-driven software architecture based testing. These include checking the internal architecture consistency, and code to architecture conformance. The distribution of the primary studies over these concerns is shown in Fig. 4. It appears thus that the majority of the motivation for MDABT is the conformance checking of the code with the architecture. Hereby, typically the architectural relations and constraints are derived and test cases are generated accordingly. These test cases are then applied on the real code to identify the possible differences. It should be noted that architecture conformance analysis is one of the key approaches for analysing the so-called architecture drift problem [19,26]. When discussing the results of RQ2 we will elaborate on the addressed concern per study and the corresponding details.

RQ2: What are the proposed solutions for model-driven architecture-based testing?

Architecture based testing (ABT) is a testing approach exploiting architectural models for testing the software system. The generic process for ABT is shown in Fig. 5. This process model or "pattern" is extracted from the thoroughly analyzed studies that is involved in this literature review. In the following we will first explain the generic process and then discuss each of the approaches that are instantiations of this generic process. Based on the figure we can identify the following issues that are present for realizing MDABT:

- *Description of the architecture*

In order to use the architecture for the purposes of MBT it should be properly described using a well-defined modeling approach. The provided model can be refined to other representations for purposes of analysis.

- *Description of test criteria*

Testing is carried out based on predefined testing goals and testing criteria. For example, the criteria might be based on coverage of graph paths. It is important to specify these criteria in a well-defined format.

- *Generating test model based on architecture*

Based on the architecture and the provided test criteria the required test model needs to be generated. The generation process can be carried out in different ways and may depend on the provided representations.

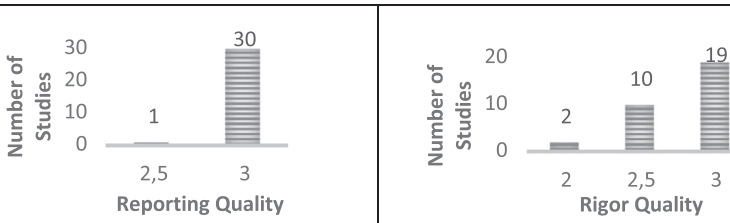
- *Test case generation based on test model*

Based on the provided test model test cases need to be generated. Different approaches might apply different generation approaches and adopt different representations for the test cases. Furthermore, test cases can be defined in multiple steps and usually a distinction is made between abstract test cases and concrete test cases.

- *Test execution*

Once the test cases have been derived these are executed on the real code or on the architecture of the system. The execution can be carried out in different ways.

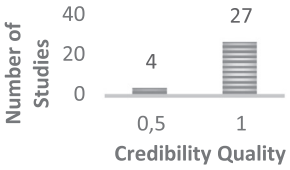
Table 5
Overall quality of the primary studies.



a) Quality of reporting of the primary studies

b) Rigor quality of the primary studies

c) Relevance quality of the primary studies



d) Credibility of evidence of the primary studies

e) Total quality of the primary studies

Table 6
Addressed concerns and studies.

Addressed concerns	Studies
Code to architecture conformity	A, C, D, E, G, H, I, M, N, K, L, O, P, Q, R, S, T, U, V, W, X, Y, Z, AA, BB, CC, DD
Internal architectural consistency checking	B, F, J, EE

- *Analysis of test results*

The final step of the process is the analysis of the test results which might be again represented in various ways. The analysis can be manual or automated.

The process in Fig. 5 is based on a general model-based testing approach. In this study we do not focus on models in general but focus on architectural models for supporting model-based testing. In principle, different ABT approaches can be derived from the same generic process. In the following we discuss the proposed solutions that we could derive from each primary study. For this, we adopt and instantiate the reference process model as discussed in Section 2. Table 7 summarizes the results of the above findings. In the following we describe each study in detail.

Study A

This study presents the adoption of chemical abstract machine (CHAM) specifications to represent software architecture and derive test cases from these specifications using coverage criteria. The testing is performed for checking the conformance of the implemented system with the specified SA. The main motivation of this paper is to use SA specifications for integration testing of the implemented system. Test

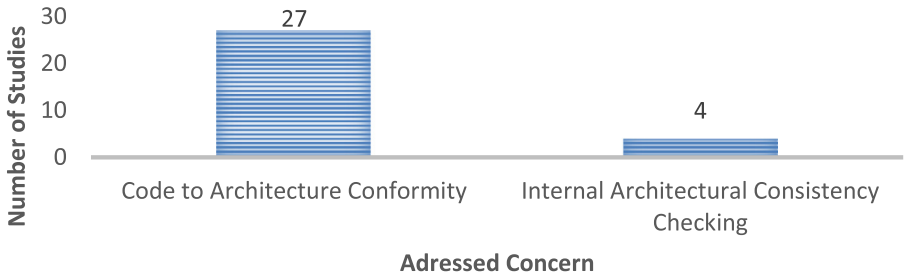


Fig. 4. Addressed concern distribution of primary studies.

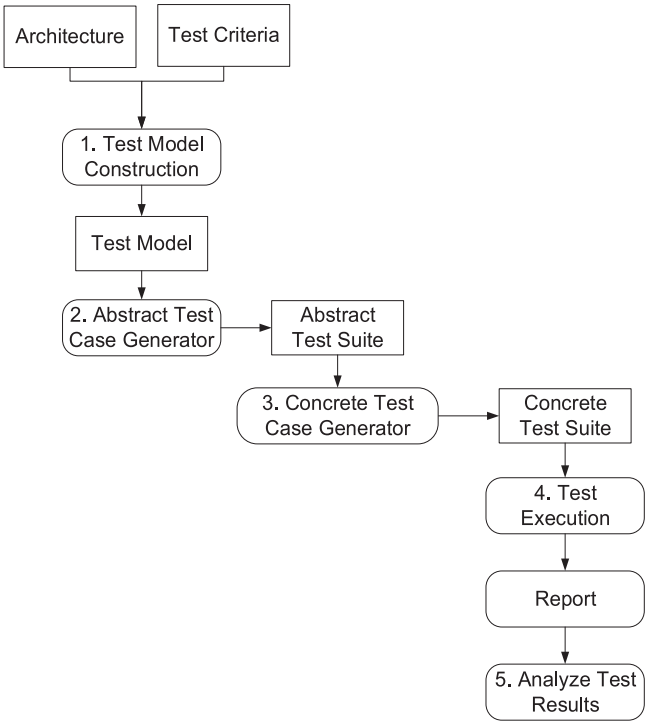


Fig. 5. Generic process for architecture-based testing.

model is generated using CHAM specifications and coverage criteria which is a transition graph called labelled transition system (LTS). From this graph, abstract labelled transition systems (ALTS) are obtained simply by applying obs function. Obs functions are functions that excludes unnecessary details for the selected view of the software architecture. Test cases are generated using the paths from ALTS graph. Each path can correspond to many concrete test cases. Test cases are generated manually by software architect. Test execution and test analysis are handled manually by software architect.

Study B

In this study the testing is done at the architecture design level to test functional properties of SA. The key concern is to check the internal architecture consistency. Hereby SA is specified using Wright ADL and six test criteria are defined based on data flow reachability, control flow reachability, connectivity and concurrency. These criteria are used as functional properties of SA to be tested and verified. In this study test model is based on behavior graph (BG) and obtained by transforming Wright ADL specifications into BG using coverage criteria as the test criteria. From the BG test model test paths are generated using the tool they created called ABaTT. Each test path is manually transformed into test cases. The test cases and test case results are automatically handled.

Study C&D&H

The key concern of these studies is architecture conformance checking. These studies present the ongoing work of previous authors in A by replacing their SA specification of CHAM model with a finite state process (FSP) model. The provided reason to use FSP instead of CHAM model is that FSP algebra is easier to map to LTS graph. Hereby, the work of the software architect is explained to generate test cases and execute them manually. UML Stereotyped Sequence Diagrams are used for filling the abstraction gap between the SA and implementation. For each architecture level sequence diagram, the software architect defines code level sequence diagrams. Global sequence diagram is obtained by combining the code level sequence diagrams where it represents code scenarios that is implementing SA path that is extracted from ALTS graph. Software architect then runs the code to see if the created sequence diagram is implemented by the system.

Study E

This study uses the same methodology explained in study D. The key concern is the conformance checking of the code to the architecture. A model checking tool is presented for software architecture using checking architectural model consistency (CHARMY) framework. SA is specified using CHARMY specifications rather than FSP model. Testing criteria chosen as coverage criteria which is selected as sub-system identification in CHARMY framework. The test model that was LTS now consists of Promela model, linear logical temporal (LTL) Formulae and Buchi Automata. The results are used by test generator engine to create test cases automatically at the SA level. The test execution and test analysis are handled manually.

Study F

The main motivation of this study is to validate architectural units using object oriented models. The study presents SA specifications in UML state diagrams, UML sequence diagrams and UML component diagrams for transforming into test model based on LTS called basic language of temporal ordering specifications (LOTOS). Basic LOTOS model is combined with test purposes to generate input/output labelled transition systems (IOLTS) model where test cases are generated using test generation using verification techniques (TGV) tool. Test cases generation and test result analysis is automated in this study.

Study G

In this study the addressed concern is again code to architecture conformity. The main motivation of the paper is to perform validation at different abstraction levels of system under test using system goals. Hereby, system behavior is implemented by a system level plan that consists of sequence of goals each describing the interactions between the components. In addition, component level plan shows the sequence that component must achieve in that scenario. The system level plan and component level plan together provide the specifications for SA. Test model is the annotated code itself. There is no test case generation the scenario is given in plans which consists of goals where assertions are defined. As the program executes the goals are emitted using rule-based recognizer and plans are tried to be matched. Test execution is automated as the tests are executed during scenario running on the program. Test analysis details are automated as the program executes it matches the component level plan.

Study I

The addressed concern of the study is code to architecture conformity. The approach uses the architecture analysis and design language (AADL) to specify SA which is then transformed into Uppaal Model (timed automata) using the coverage criteria as the test criteria. Using the set of automata paths, consistency and completeness check is performed by applying model checking. Moreover, automata paths are translated into concrete test cases using a mapping between architecture specification and implementation. At last test execution and analysis is automatically handled.

Study J

The main motivation of this paper is to use SA in model based testing to detect defects earlier in software lifecycle. Testing is performed at architectural level. The study presents the use of hierarchical predicate transitions nets (HPrTNs) model to explain the behaviors of SA obtained by transforming Acme ADL specifications with coverage criteria as the test criteria. The HPrTNs model is divided into sub graphs which are abstract models depending on the model. Later using sub graphs architecturally significant path are extracted and test cases are generated from the extracted paths. Test execution and analysis is performed automatically.

Study K

The addressed concern of this study is code to architecture conformity. This study presents the application of SA based testing methodology to service oriented applications in distributed systems. The service composition which is our architecture in this case is expressed using business process execution language (BPEL). In the SA specifications architecture topology and message exchange knowledge is included. The extended control flow graph (ECFG) test model is used to

Table 7
Overview of the identified approaches.

Study	Criteria	Architecture Model	Test Model Construction	Test Model	Abstract Test Case Generation	Abstract Test Suite	Concrete Test Case Generation	Concrete Test Suite	Test Case Execution	Test Oracle
A	CC	CHAM	M	LTS	M	ALTS	M	ALTS Paths	M	M
B	CC	Wright ADL	A	BG	NA	NA	M	BG Paths	A	A
C	CC	FSP Model	A	LTS	M	ALTS	M	ALTS Paths	M	M
D	CC	FSP Model	A	LTS	M	ALTS	M	ALTS Paths	A	A
E	TPM	UML	A	Promela	M	NA	M	Promela Paths	M	M
F	TPM	UML	A	IOLTS	A	Test Graph	A	Tree and Tabular Combined Notation	A	A
G	TPM	GoalML	A	Annotated Code	NA	NA	M	Annotated Code	A	A
H	CC	FSP Model	A	LTS	A	ALTS	A	ALTS Paths	A	A
I	CC	AADL	A	Uppaal	NA	NA	A	Uppaal Paths	A	A
M	CC	UML	A	Test Graph	A	Test Graph Paths	A	Cucumber Scenarios	A	A
N	CC	UML	A	U2TP	NA	NA	A	JUnit	A	A
J	CC	Acme ADL	A	PetriNet	A	HPrTNS	A	Petri Net Paths	A	A
K	CC	BPEL	A	ECFG	A	CFG	A	CFG Paths	A	A
L	CC	UML	A	eDelta Models	A	xUnit	A	JUnit	A	A
O	CC	LTS	A	ALTS	NA	NA	M	ALTS Paths	A	A
P	CC	UML	A	xUnit	NA	NA	A	JUnit	A	A
Q	CC	Wright ADL	M	ACFG	A	ACDG	A	ACDG Paths	A	A
R	CC	UML	A	PetriNet	NA	NA	A	Petri Net Paths	A	A
S	CC	UML	A	UTP	NA	NA	A	Cunit	A	A
T	CC	MSC	A	MSC	NA	NA	A	MSC Test Cases	M	M
U	CC	UML	A	UTP	NA	NA	A	CPPUnit	A	A
V	CC	CDM	A	DG	NA	NA	A	TUM	A	A
W	CC	UML	A	UTP	NA	NA	A	TTCN-3	A	A
X	CC	EFSM	A	EFSM -SeTM	NA	NA	A	EFSM -SeTM Paths	A	A
Y	CC	UML	A	Activity Diagram	NA	NA	A	Activity Diagram Paths	A	A
Z	CC	UML	A	PetriNet	NA	NA	A	Petri Net Paths	A	A
AA	CC	UML	A	UTP	A	xUnit	A	JUnit	A	A
BB	CC	UML	A	VDM-SL	NA	NA	A	VDM	A	A
CC	CC	WSDL-S	A	ESG	NA	NA	A	ESG Paths	A	A
DD	CC	PrT	A	PetriNet	A	Transition Tree	A	JUnit	A	A
EE	CC	UML	A	DERCS	NA	NA	A	DERCS	A	A

CC: Coverage Criteria; TPM: Test Purpose Matching; A: Automatic; M: Manual; NA: Not Applicable.

generate test cases by transforming BPEL specifications with coverage criteria into the test model. ECFG consists of control flow graphs (CFG) and from each CFG test paths are derived. Test cases are executed and results are analyzed automatically.

Study L

The addressed concern of this study is code to architecture conformity. This study presents specification of SA is using UML state, sequence and component diagrams. UML state machines are used for representing component behaviors of SA. All UML models are combined into one test model consists of state machine test model, message sequence chart test model and component test model. Test cases are generated from state machine test model are executed on those components which means that testing is done at architectural level. Test cases generated from the message sequence chart are executed in the system which implies that test cases are executed at code level system. Test execution and analysis is automatically handled.

Study M

The study addresses the concern of conforming given architecture to implementation. In this study, a tool for generating behavior driven tests using behavioral UML diagrams are proposed. Software architecture is defined in terms of different behavioral UML diagrams such as use case, interaction, activity diagrams and using full graph coverage as

coverage criteria tool creates so called model test graph. Each path of the test graph represents a concrete behavioral test case scenario which is executed automatically via another tool called Cucumber. Testers then writes mappings for each generated Cucumber scenarios and execute tests.

Study N

This study concerns the correctness of the code with respect to the architecture. An approach is proposed for model driven automated tests generation using UML sequence diagrams as software architecture. UML sequence diagrams together with coverage criteria is used for creating test models as U2TP sequence diagrams. The generated platform specific models are then transformed into test cases which are then executed and analyzed on small example for validating the proposed approach.

Study O

The study aims to check the conformance of the code with respect to the architecture. Hereby an LTS is defined as an architecture model and coverage criteria as test criteria for creating a test model ALTS. There is no abstract test model for this approach each path of ALTS model is transformed in to concrete test case where executed and analyzed automatically.

Study P

This study aims to check the code to architecture conformity. UML sequence diagrams are used to represent SA where the proposed approach automatically creates xUnit model as a test model for generating concrete test cases based on xUnit such as jUnit, cUnit, cppUnit etc. For demonstration purposes, the approach generates jUnit test cases which can be automatically executed and results are analyzed.

Study Q

This study introduces a new algorithm for slicing of architecture and checking the conformance of architecture to code. The authors represent SA using Wright ADL. Test model of architecture control flow graph (ACFG) is manually created using coverage criteria. This model is automatically transformed into abstract test model of architecture control dependence graph where each path of this abstract test model is then mapped to concrete test cases automatically. Generated test cases can be executed and the results analyzed automatically.

Study R

In this study UML models are used to generate test cases to ensure the conformity of distributed systems architecture with respect to each other based on scenarios modeled in sequence diagrams. SA model is transformed into timed event-driven colored petri nets (TEDCPN) using coverage criteria for all models. Each path of the test model is then transformed into concrete test cases which are executed and analyzed automatically.

Study S

This study S addresses aims to check the conformity of code to the given architecture to catch the defects at earlier lifecycle of development in real-time critical embedded systems.

The main motivation of the paper is to elevate a newly proposed testing architecture for embedded systems. The SA is presented using UML state diagrams and coverage criteria is selected as test model creation criteria. Test model of type UML testing profile (UTP) models are generated. Concrete test cases are generated using UTP models in the form of cUnit. Generated test cases can be automatically executed and analyzed.

Study T

This study S addresses aims to check the conformity of code to the given architecture. This study greatly differs from the others by the model that is used for SA model and test model which are basically the same model type which is message sequence charts (MSC). MSC is a specialized type of sequence diagrams. Coverage criteria is used for generating MSC test model where each path along the chart represents a concrete test case. Concrete test cases are manually executed and analyzed.

Study U

In this study test cases are generated for checking the conformance of architecture to code for critical properties. In this study, SA is represented using UML models. UTP test model is generated using SA model along with coverage criteria. UTP test models are then translated into CPPUnit concrete test cases which can be automatically executed and analyzed.

Study V

This study represents the dependencies between components and checks the dependency relations at the implementation level for conformance to architecture. SA is represented using component dependency model (CDM) which is then transformed into dependency graph (DG) using coverage criteria. Time usage models (TUM) are generated from DG models as concrete test cases.

Study W

In this study test cases are generated for conformance checking of code level implementation to architecture models for software-intensive mission-critical systems. SA is represented using UML models and along with coverage criteria UTP test models are generated. Test models are transformed into testing and test control notation (TTCN-3) test cases which are automatically executed and analyzed.

Study X

This study aims to check the conformance of web service components and the given architecture. SA is represented using extended finite state machine (EFSM). Test model of EFSM–sequence test model (EFSM–SeTM) is generated using coverage criteria. EFSM–SeTM is a model in which EFSM is extended by sequence diagram.

Study Y

In this study the authors check for the conformance of the camera software application for mobile phones with respect to given UML architecture of the software. SA is represented using UML models. Along with coverage criteria activity diagram test model is generated. Each path of the diagram corresponds to a concrete test case to be executed. Test execution and result analysis is performed automatically.

Study Z

In this study, the authors generate tests cases in the form of petri net paths to check the conformance of implemented system with respect to architecture. SA is represented using UML class, collaboration and state diagrams. Along with the coverage criteria these architecture models are transformed into concurrent object-oriented petri nets (CO-OPN) test models. Each path on CO-OPN model corresponds to a concrete test case to be executed on system under test. Test execution and result analysis are performed automatically in the proposed approach.

Study AA

In study AA, the authors generate test cases to conform the implemented system with respect to UML models of software. SA is represented using UML sequence diagrams. Architecture model and coverage criteria as the test criteria generates UTP test model. Abstract test cases of xUnit based models are generated from UTP test models. JUnit test cases are automatically derived from xUnit models in which test execution and result analysis can be performed automatically.

Study BB

This study presents an approach to check the conformity between provided architecture models and implemented systems of distributed systems. SA is represented using UML sequence diagrams. Test model of type Vienna development method specification language (VDM-SL) is generated using architecture model and coverage criteria. Concrete test cases are generated using the specifications provided in the test model. Test case execution and result analysis is performed automatically.

Study CC

In this study test cases are generated for services deployed in cloud computing environment with provided service as a software platform. The key concern is again conformance checking. Web service architectures are represented using web service semantics (WSDL-S) which is an extension of web service description language (WSDL). Event

sequence graphs (ESG) are generated using WSDL-S model and coverage criteria. Each path on ESG is transformed into concrete test cases which can be executed automatically. Test case execution results are then automatically derived.

Study DD

In this study test cases are generated to check the conformance of implementation and given models. SA is represented using predicate/transition (PrT) nets. Proposed approach automatically generates test models based on colored Petri Nets using coverage criteria and SA models. Abstract test case in form of Transition Trees are generated which are then transformed into JUnit test cases. JUnit test cases are executed automatically and results are analyzed.

Study EE

In this study EE, an approach is presented to generate test cases to validate and verify the correctness of designed embedded system architecture. SA is represented using behavioral UML diagrams which are: use case models, interaction diagrams, sequence diagrams, collaboration diagrams and state diagrams. Distributed Embedded Real-time Compact Specification models are generated as test models with coverage criteria and provided architecture models. Concrete test cases are generated from the test models are same of type and can be executed automatically. Results of the test case executions are automatically analyzed.

4.4.1. Summary

So far, we have shown the different MDABT approaches which we explained as instantiations of the generic reference model as shown in Fig. 5. On one hand we can state that all these approaches share the similar approach since these could all be discussed based on the reference process. On the other hand, each of these approaches were different for the different parts of the reference process. The reference process helped not only to understand each of these processes but also was a useful instrument to compare the proposed approaches. Although MDABT can be applied to any domain, some domains had a clear interest including embedded systems, real-time systems, cloud services, distributed systems, mission-critical systems and large-scale variant-rich systems.

Obviously, the SA specification appeared to be an important part of our process model and different approaches applied different modeling approaches. Fig. 6 shows the distribution of the various SA specifications over the approaches from the primary studies. As it can be observed from the figure, UML is the most preferred approach in the literature. FSP model follows UML as the second most frequently used approach. The main reason that UML is the most preferred approach could be attributed due to the fact that it is a popular approach in system modeling. Hence, the learning curve for UML is lower and the willingness to adopt UML models for testing is probably higher.

Based on the reference process, the second differing aspect in the approaches is the adopted testing criteria in which most of the studies adopts coverage criteria for testing purposes. A small proportion of the studies use test purposes that are exclusively defined for the approaches presented. Fig. 7 shows the distribution of testing criteria over the selected studies. Coverage criteria applies for graph based test models in

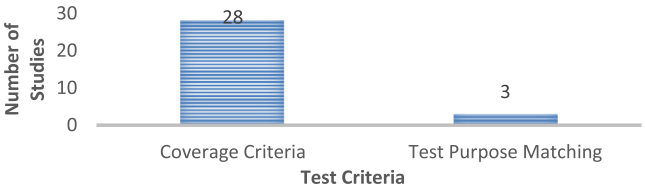


Fig. 7. The identified test criteria.

which all the paths of the graph denotes a test case. Based on this we can conclude that most of the studies adapted graph based approaches and test criteria.

Another important aspect of the process model is test models, which are used to generate test cases. Fig. 8 shows the test model distribution over the various approaches. LTS, PetriNet, and UTP appeared to be the most preferred test models in the literature. Most of the studies use graph-based or automata-based test models where test cases are generated from extracted paths from nodes to other nodes. In one of the studies SA is specified using a domain specific language GoalML and no test models are used. Hereby, the tested properties are annotated in the implementation itself and during the execution of the system the assertions are emitted.

Fig. 9 shows the distribution of the identified test model construction approaches. Most of the studies apply automated generation of test models, but in two of the studies this process is executed manually. Manual test model generation is a both effort and resource intensive process which can be manageable for small scale applications. However, for large-scale applications this does not appear to scale well.

Another important aspect of the reference process model is the abstract and concrete test suite generations. Fig. 10 shows the distribution of the approaches for abstract test model types for the given studies. From the figure we can see that most of the studies do not have any abstract test suites as represented by the label “NA” (not applicable). Except two of the studies, all the studies use graph-based notation (mostly ALTS) for abstract test suites. Hence, we can infer that graph-based solutions are popular among the primary studies.

Fig. 11 shows the distribution of the abstract test case generation approaches. The majority of the studies (19) did not provided an abstract test case generation approach as indicated by the label “NA”. For the ones that did apply it, we can see that an automated generation process is mostly adopted. As in the test model creation, manual construction of abstract test case generation is usually not preferred since it is effort and resource consuming.

Fig. 12 shows the distribution of the adoption of concrete test suite types. Here, we can we can see that ALTS and JUunit test cases are the most frequently used approaches. From the overall perspective, graph paths based notations for concrete test cases are dominating the studies but in later studies we see that xUnit based concrete test suites are preferred.

Fig. 13 shows the distribution of the concrete test suite generation methods. Here we can observe that automated approaches are most frequently used to create concrete test suites from abstract test suites. However, there are still manual approaches present for the concrete test

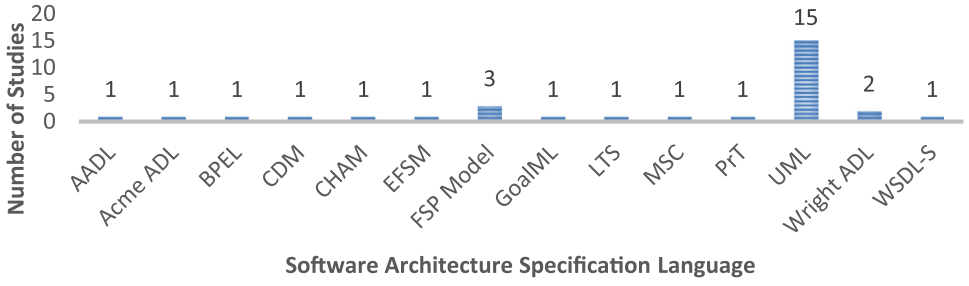


Fig. 6. The identified architecture description approaches and the frequency over the different studies.

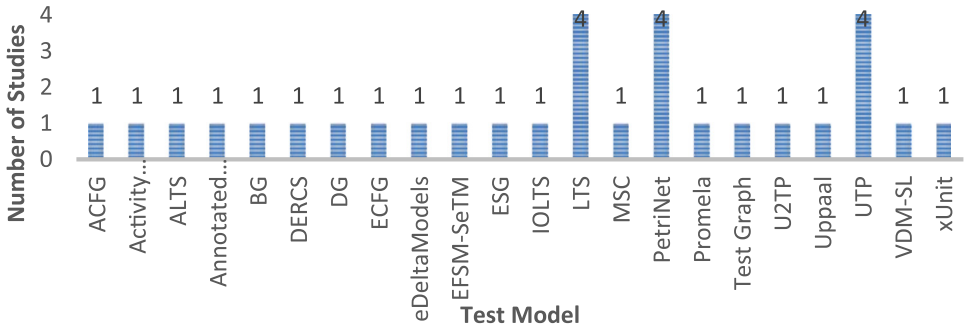


Fig. 8. The identified test model representations.

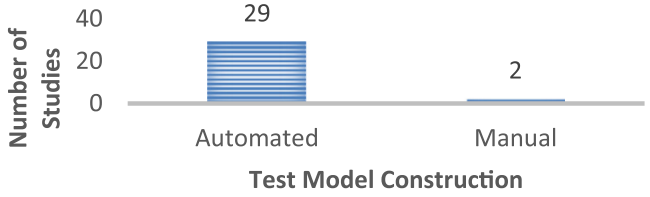


Fig. 9. The identified test model construction.

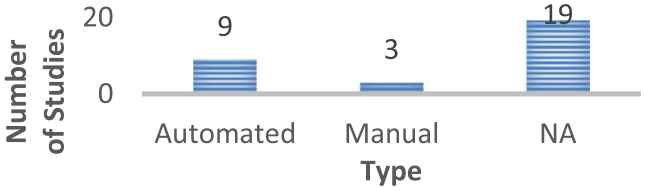


Fig. 11. The identified abstract test case generation type.

case generation.

In Fig. 14, the distribution of test case execution over the studies can be seen. Most of the studies again use automated test case generation whereby testing is carried out at the SA level rather than code level. In manual test case development, we could observe both SA level and code level testing. Manual test case execution is simply executing the concrete test case on the code or SA, which will take lots of effort for large test suites.

Fig. 15 shows the distribution of test analysis types, which considers the analysis of the executed tests. Most of the studies used automated analysis where the result of failing or passing is automatically determined. In manual cases reporting of each test case execution will be handled manually and for large test suites it is unfeasible for most cases.

RQ3: What are the existing research directions within model-driven architecture based testing?

After we have characterized each model-driven architecture-based testing approach we will now discuss the existing research directions. The research directions are on the hand derived from the identified concerns and solution directions for applying MDABT, on the other hand these have been sometimes provided independently. For RQ1 we have identified the two key concerns including (1) code to architecture conformity and (2) internal architecture consistency checking. For RQ2 we have identified 31 approaches each of which had different properties. Table 8 presents the overview of the identified research directions. In the following we describe the indicated research directions in the primary studies.

- *Need for executable models for representing architecture*

For supporting model-based testing several authors have indicated the need for precise models that can be processed by model compilers. This has been indicated for both the concerns of code to architecture conformity and internal architecture consistency checking (RQ1).

Most of the research in this domain is adopted from the developments in model-driven development in which software language engineering plays an important role. Different solutions have been proposed (RQ2) in the studies using different representation techniques for SA specifications. For supporting model-driven architecture-based testing it is important that the architecture models are at the level of executable models. This typically requires that the models are expressed in some domain specific language.

- *Abstraction difference between architecture and code*

This research direction was mainly identified in the studies that focus on conformance checking of the code with respect to the architecture (RQ1). As stated before, these were the majority of the studies. The adoption of architecture models for supporting model-based testing provides both an abstract view of the system and the actual implementation of the system. Creating tests at the SA level appeared to be simpler than creating tests for code level that are derived from SA specifications. Based on the provided solutions (RQ2) it appeared indeed that a challenge was to fill the gap between the SA and the

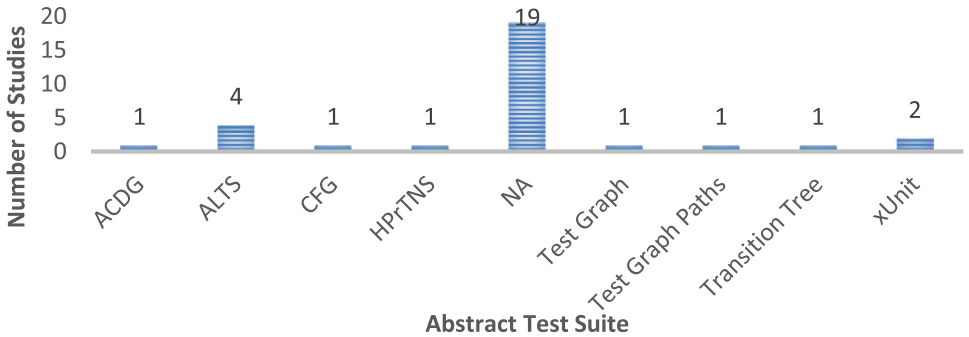


Fig. 10. The identified abstract test suite.

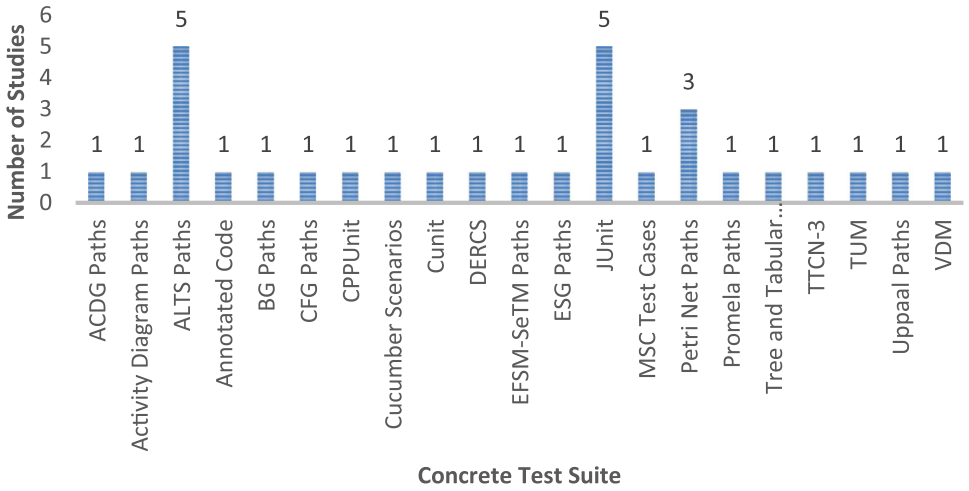


Fig. 12. The identified concrete test suite.

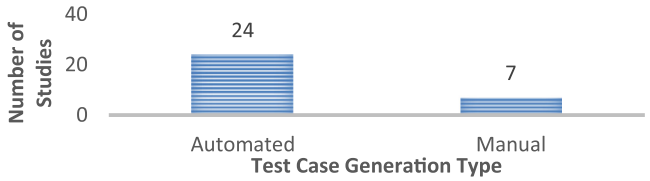


Fig. 13. The identified concrete test case generation type.

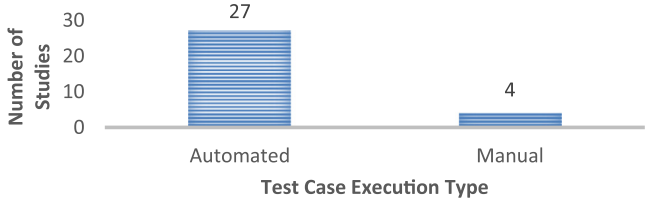


Fig. 14. The identified test case execution.

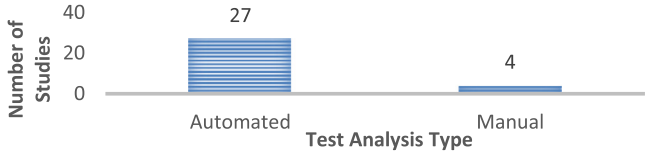


Fig. 15. The identified test analysis type.

Table 8
Overview of the identified obstacles that has been discussed in the related primary studies.

Research direction	Discussed in primary study
Need for executable models for representing architecture	A, B, C, D, E, F, H, J, L, P, Q, R, S, T, U, V, W, Y, Z, BB, CC, DD, EE
Abstraction difference between architecture and code	A, B, C, D, E, F, G, H, J, O, S, AA
State explosion problem	A, C, D, F, H, M, K, L, O, Q, S, V, X, Y, Z, DD
Architecture views	C, D, E, O
Automating the process by tools and cost-benefit analysis	A, B, C, D, F, G, , H, I, M, N, L, O, P, Q, R, S, T, U, V, W, X, Y, Z, AA, BB, CC, DD, EE
Applying MDABT on complex real systems	A, B, C, D, F, G, H, I, M, N, J,K L, O, R, W, X, AA

implemented system. Although several solutions have been provided such as architectural styles, this problem has been identified in many studies as one of the key challenges.

- *State explosion problem*

In testing and model-based testing the generation of test cases easily leads to a combinatorial explosion and becomes less tractable for the human engineer. This appears also to be the case for model-driven architecture-based testing, and has been addressed in both the studies that concern conformance checking and internal architecture consistency (RQ1). Although the adoption of the presented model-driven solutions (RQ2) supports the automated generation of test artifacts this intractability problem has also been pointed out by several authors. For example, when generating test cases from test model that is based on graph or automata, full path coverage criteria causes numerous test cases. Moreover, some studies have used model checking integrated with SA based testing where the results from model checking are used for generating test cases. Yet, this challenge of state explosion problem seems to be recurring and there seem still to be many challenges for finding more effective criteria in the test generation process.

- *Architecture views*

As discussed in the background, architecture needs to be modeled using multiple architecture views to represent the various stakeholder concerns [5,25]. This is again applicable both for conformance checking and internal architecture consistency analysis (RQ1). For conformance checking it appeared to be important to identify the proper architecture views on which the code would be checked. For internal architecture consistency it seemed to be important to check the consistency among the views. The presented solution approaches (RQ2), however, largely tend to focus on a single view of the architecture which is usually represented using graph formalisms. A few authors have indicated the need for representing views on the graphs which are more like queries on a single representation format. These approaches however do not adopt the current notion of viewpoints that adopt different perspectives on the architecture based on different stakeholders. Integrating architecture views, as such, will be a necessary enhancement to the existing architecture-based testing approaches.

- *Need for automating the process and its cost-benefit analysis*

Both conformance checking and internal architecture consistency processes become less tractable and require automation for large scale systems (RQ1). While analyzing the provided solutions (RQ2) we could identify that the MDABT process as we have presented is largely followed. Further we can conclude that automation has been provided for these steps. Yet, no single study has presented an approach that

completely automates the overall process. This has also been identified and discussed by the authors of these studies. Further automation of the overall process seems to be a recurring future work. Another important issue that seems to be present in the current literature is the lack of justification and the cost-benefit analysis of the MDABT. Some progress can be observed in this context, however none of the studies have adopted a systematic approach to evaluate and justify the benefits of the automation process. This has been also explicitly addressed as a challenge by many authors.

- *Need for applying MDABT on complex real systems*

It is very important to use empirical validations while assessing your methodology. In most of the studies the proposed solution approaches (RQ2) are assessed on simple case studies such as simple client server applications. On the other hand, some studies used industry cases to apply their methodology which are more complex and already in use. Nevertheless, there is still a need for applying MDABT on more complex industry cases to justify the effectiveness of these approaches. This applies for both the concerns of conformance checking and internal architecture consistency (RQ1).

4.5. Threats to validity

Every SLR is susceptible to various validity threats, which have to be addressed for the results to be acceptable. In the following we describe the possible validity threats and discuss the adopted risk mitigation strategies.

One of the primary dangers to legitimacy of SLR is the publication bias. Publication bias can be explained by researcher's trend to publish positive results rather than negative results of their studies. It is proposed in [12] to perform research protocol with research question to manage publication bias as we have done in this study. Later in the study we identified search scope, search method and constructed search string for different publication platforms to query on the MDABT area. Incompleteness caused by search keywords is another danger for legitimacy of SLR. To mitigate this risk, we have constructed keyword list in a repeated manner by pre-performing sample searches in the domain. The construction of the keyword list was handled manually by adding new keywords if we were not able to retrieve related studies to our field. As stated before, the constructed search strings for different platforms are located in Appendix-A.

Even though we have carefully designed the search queries there is still a threat that we can still not find related studies such as technical report, theses and company journals. In our study these neglected studies can have importance in terms of completeness and validation by strong case study. In our SLR we did not include such studies. Yet another danger to legitimacy of SLR can be seen the limited functional operations of publication platforms. Publication platforms have limited ability to perform complex queries on the database such as the length of the query and retrieving unrelated studies. As a result of this, study inclusion and exclusion criteria are defined and studies are filtered manually with respect to created criteria. After filtering and selecting the final set of primary studies we extracted data from the primary studies. We have constructed data extraction form to systematically extract data from the studies by reading each study according qualitative and quantitative properties.

5. Related work

Souza et al. [23] provide a mapping study to characterize the state of the art on how software architecture and software testing have been explored in a complementary way. The systematic mapping study identified 27 primary studies that have explored the use of information related to software architecture to support the testing activity. The identified approaches have covered the integration, unit, and regression phases, and also considered functional and structural techniques. The basic conclusion of the paper is that there are still a range of open research topics to be explored, in particular, tool support. In contrast to this mapping study we have provided a systematic literature review and have analyzed the primary studies in detail. We have also provided an explicit and detailed list of the addressed concerns, the proposed solutions and the research directions.

Rafi et al. [22] provide a systematic literature review and survey on benefits and limitations of automated software testing. The SLR selects 25 studies and conduct data extraction for both benefits and limitations of automated software testing. The data extraction results are on its turn used to conduct a practitioners' survey for determining whether the benefits and limitations are of relevance for practitioners. The survey has been performed with 115 practitioners. The authors conclude that the main benefits of test automation are reusability, repeatability, effort saved in test executions, and improvement of the test coverage. Our systematic literature review is also in the scope of automated software testing, but we have in focused directly on architecture-driven model based testing. For this we have provided more dedicated insight which is useful for paving the way for further research in this domain.

Neto et al. [20] 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 analysis 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. This study is similar to our study in that it considers model-based testing in general. However, it does not explicitly consider the adoption of architecture models for generating test cases.

Gurbuz et al. [9] have focused on the adoption of model-based testing for safety-critical systems. Detecting the potential faults is crucial for these systems since a failure or malfunction may result in death or serious injuries to people, equipment or environment. Again, hereby one of the key challenges is the derivation of test cases that can identify the potential faults, and MBT can be useful in this domain as well. The paper provides a systematic review to identify the state-of-the-art model-based testing for software safety. In the mapping study 604 papers have been identified and 32 of them have been selected as primary studies to analyze the benefits of MBT for software safety. The study shows that although MBT can provide important benefits for software safety testing still a number of challenges must be overcome to support reliable model-based testing approach for safety. The study of Gurbuz et al. have a specific focus, that is, model-based testing for software safety. We focused on architecture-driven model-based testing and did not consider a specific quality factor.

Felderer et al. [6] provide a taxonomy for model-based security testing approaches. The taxonomy is based on a comprehensive analysis of existing classification schemes for model-based testing and security testing. The study has identified 119 publications on model-based security testing and classified these according to the defined filter and evidence criteria. Likewise, the article provides an overview of the state of the art in model-based security testing and discusses promising research directions with regard to security properties, coverage criteria and the feasibility and return on investment of model-based security testing. Here again, the focus is on adopting model-based testing for a particular quality concern, which was not our focus.

Besides of testing, software architecture has also been used for model-checking. Model checking is a formal verification technique that aims for the automated analysis of a systems' behavior with respect to selected properties. It takes as input a system model and a property specification[2,3]. The output is "true", or false with a counter-example when an error is detected. Zhang et al. [29] provide a classification and

comparison of model checking software architecture techniques. The authors describe the main activities in a model checking software architecture process. Then, a classification and comparison framework is provided which is subsequently used to compare model checking software architecture techniques. Similar to software testing, model checking is one of the verification approaches. The study did not apply a systematic literature review but the outcome of the study could be considered complementary to our study in that it focuses directly on the adoption of architectural descriptions in the verification process.

To sum up, we can state that none of the above studies have adopted architecture descriptions in the model-based testing process. On the other hand, there are interesting complementary issues such as the focus on particular quality concerns and the adoption of model-checking.

6. Conclusion

In general, exhaustive testing is not practical or tractable for most real programs due to the large number of possible inputs and sequences of operations. As a result, selecting the set of test cases which can detect possible flaws of the system is the key challenge in software testing. MBT aims to automate the generation and execution of test cases using models based on system requirements and behavior. MBT provides several important advantages such as improved test coverage and fault detection, reduced testing time, and reduction in cost. So far, various different models have been adopted to derive the test cases in MBT. In this study, we have focused on adopting architecture models to provide automated support for the test process leading to the notion of model-driven architecture-based testing (MDABT).

Different from other models such as finite state machines, which are typically at the detailed design level, software architecture provides a gross-level representation of the system at the higher abstraction level. To depict the state-of-the-art on MDABT, to identify the current solutions as well the open research challenges, we have carried out a systematic literature review on MDABT. We reviewed 731 papers that are discovered using a well-planned review protocol, and 31 of them were assessed as primary studies related to our research questions. Considering the different studies and the background on model-based testing in general we derived a reference process model to depict model-based testing. Based on our study we can derive several important lessons.

First of all, as we have derived from the primary studies the basic motivation for adopting MDABT seems to be the conformance checking of the code with the architecture. Hereby, typically the architectural relations and constraints are derived and test cases are generated, which are then applied on the real code. The motivation for checking the internal consistency of an architecture using MDABT was identified in a few primary studies. Although MDABT can be applied to any domain, some domains had a clear interest including embedded systems, real-time systems, cloud services, distributed systems, mission-critical systems and large-scale variant-rich systems.

Regarding the provided solutions we can state that all the presented approaches in the primary studies could be considered as instantiations of the reference process model. This shows that there is a consensus on the generic model-based testing process. The approaches differ in the description of the architecture models, description of test criteria, the generation process of the test models, the test execution and the analysis of the results. We have discussed each of these elements in detail. Besides of the various usages of the process elements, some approaches in the literature also relax the control flow that is defined in the generic process model. This is typically done by deriving test cases directly from a concrete test suite and thus ignoring the generation of an abstract test suite.

Further, although each approach appears to be interesting and effective for a given scope several obstacles still need to be solved. We have discussed these obstacles as the need for executable models for representing architecture, coping with the abstraction difference between architecture and code, the combinatorial state explosion problem, the lack of architecture views to consider multiple different concerns, the need for automating the process by tools, the cost-benefit analysis, and the need for applying MDABT on complex real systems to provide evidence of the approaches with respect to conventional approaches. Some of these obstacles are an inherent problem also in the broader domain of MBT but these all seem to be relevant obstacles for MDABT in particular.

Finally, from our study we can state that the potential of MDABT has not been fully exploited yet. Given the reference process model that we have provided and the insight in the various different approaches, we can derive many more different possible MDABT processes than the 31 approaches that we could derive from the literature. It would be worthwhile to explore these to optimize the benefits for MDABT in the future research. Similar to automated software testing in general, the MDABT approach could be used also for different quality concerns such as safety and security. Given the many benefits of MBT it will be worthwhile to elaborate on these in further research and likewise pave the way for more effective automated testing in practice.

We believe that the results of this study can be used by researchers as an input to advance the research on MDABT and derive even more effective solutions that can be applied in practice. Our own future research will be in alignment with the results of this study and consider the various obstacles to derive novel MDABT solutions. Furthermore, we aim to apply MDABT for large scale industrial case studies.

Appendix-A. Search strings

Electronic Database	Search String
IEEE Xplore	(("Document Title":"model based testing" OR "Document Title":"model based software testing" OR "Document Title":"model-based testing" OR "Document Title":"model-based software testing" OR "Document Title":"model driven testing" OR "Document Title":"model driven software testing" OR "Document Title":"model-driven testing" OR "Document Title":"model-driven software testing" OR "Document Title":"model based test" OR "Document Title":"model based software test" OR "Document Title":"model-based test" OR "Document Title":"model-based software test" OR "Document Title":"model driven test" OR "Document Title":"model driven software test" OR "Document Title":"model-driven test" OR "Document Title":"model-driven software test") AND ("Document Title":"architecture")) OR ("Document Title":"architecture based testing" OR "Document Title":"architecture based software testing" OR "Document Title":"architecture-based testing" OR "Document Title":"architecture-based software testing" OR "Document Title":"architecture driven testing" OR "Document Title":"architecture driven software testing" OR "Document Title":"architecture-driven testing" OR "Document Title":"architecture-driven software testing" OR

	"Document Title":"architecture based test" OR "Document Title":"architecture based software test" OR "Document Title":"architecture-based test" OR "Document Title":"architecture-based software test" OR "Document Title":"architecture driven test" OR "Document Title":"architecture driven software test" OR "Document Title":"architecture-driven test" OR "Document Title":"architecture-driven software test") 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":"architecture")) OR ("Abstract":"architecture based testing" OR "Abstract":"architecture based software testing" OR "Abstract":"architecture-based testing" OR "Abstract":"architecture-based software testing" OR "Abstract":"architecture driven testing" OR "Abstract":"architecture driven software testing" OR "Abstract":"architecture-driven testing" OR "Abstract":"architecture-driven software testing" OR "Abstract":"architecture based test" OR "Abstract":"architecture based software test" OR "Abstract":"architecture-based test" OR "Abstract":"architecture-based software test" OR "Abstract":"architecture driven test" OR "Abstract":"architecture driven software test" OR "Abstract":"architecture-driven test" OR "Abstract":"architecture-driven software test")
ACM Digital Library	acmdlTitle:(+ architecture + software + test "model" "based" "driven") OR recordAbstract:(+ architecture + test + software + model + ("based" "driven"))
Wiley Interscience	(model test software architecture in Abstract) OR (model test software architecture in Publication Titles)
Science Direct	(((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 driven test) OR Title(model driven software test) OR Title(model-driven test) OR Title(model-driven software test)) AND (Title(architecture))) OR (Title(architecture based testing) OR Title(architecture based software testing) OR Title(architecture-based testing) OR Title(architecture-based software testing) OR Title(architecture driven testing) OR Title(architecture driven software testing) OR Title(architecture-driven testing) OR Title(architecture-driven software testing) OR Title(architecture based test) OR Title(architecture based software test) OR Title(architecture-based test) OR Title(architecture-based software test) OR Title(architecture driven test) OR Title(architecture driven software test) OR Title(architecture-driven test) OR Title(architecture-driven software test))) 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 driven test) OR Abstract(model driven software test) OR Abstract(model-driven test) OR Abstract(model-driven software test)) AND (Abstract(architecture))) OR (Abstract(architecture based testing) OR Abstract(architecture based software testing) OR Abstract(architecture-based testing) OR Abstract(architecture-based software testing) OR Abstract(architecture driven testing) OR Abstract(architecture driven software testing) OR Abstract(architecture-driven testing) OR Abstract(architecture-driven software testing) OR Abstract(architecture based test) OR Abstract(architecture based software test) OR Abstract(architecture-based test) OR Abstract(architecture-based software test) OR Abstract(architecture driven test) OR Abstract(architecture driven software test) OR Abstract(architecture-driven test) OR Abstract(architecture-driven software test)))
Springer	'software AND architecture AND test AND model AND "based testing" AND (based OR driven) AND NOT (hardware AND circuit)'

ISI Web of Knowledge	(((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 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 = "architecture")) OR (TS = "architecture based testing" OR TS = "architecture based software testing" OR TS = "architecture-based testing" OR TS = "architecture-based software testing" OR TS = "architecture driven testing" OR TS = "architecture driven software testing" OR TS = "architecture-driven testing" OR TS = "architecture-driven software testing" OR TS = "architecture based test" OR TS = "architecture based software test" OR TS = "architecture-based test" OR TS = "architecture-based software test" OR TS = "architecture driven test" OR TS = "architecture driven software test" OR TS = "architecture-driven test" OR TS = "architecture-driven software test"))

Appendix-B. List of primary studies

A. Bertolino, A.; Corradini,F. ;Inverardi, P.;Muccini, H.(2000). Deriving Test Plans from Architectural Descriptions. pp. 220–229.

B. Jin, Z.; Offutt, J.(2001). Deriving Tests From Software Architectures. pp. 308–313.

C. Muccini, H.; Bertonlino, A., Inverardi, P.. (2004). Using Software Architecture for Code Testing. pp. 160–171.

D. Muccini, H.;Dias, M.;Richardson,D.J. . (2004). Systematic Testing of Software Architectures in the C2 Style. pp. 295-309.

E. A. Bucchiarone, H. Muccini, P. Pelliccione, P. Pierini. (2004). Model-Checking plus Testing from Software Architecture Analysis to Code Testing. pp. 351–365.

F. Scollo, G.; Zecchini, S..(2005). Architectural Unit Testing. pp. 27–52.

G. Winbladh, K.; Alspaugh, A.T.; Ziv, H.; Richardson, D.(2006). Architecture Based Testing Using Goals and Plans. pp. 64–68.

H. Muccini, H.; Dias, M.; Richardson, J.D.(2006). Software Architecture-Based Regression Testing. pp.1–18.

I. Johnsen, A. ; Pettersson,P.; Lundqvist,K.(2009). An Architecture-Based Verification Technique for AADL Specifications. pp. 105–113.

J. Reza, H.; Lande, S.(2010). Model Based Testing Using Software Architecture. pp. 188–193.

K. Keum, C.; Kang, S.; Kim, M..(2013). Architecture-Based Testing of Service-Oriented Applications In Distributed Systems. pp. 1212–1223.

L. Lochau, M.; Lity, S.; Lachmann, R.; Schaefer, I.; Goltz,U..(2013). Delta-oriented Model-Based Integration Testing of Large-Scale Systems. pp. 63–84.

M. Li, N.; Escolana, Anthony.; Kamal, T.;(2016). Skyfire: Model-Based Testing With Cucumber. pp. 393–400.

N. Elallaoui, M.; Nafil, K.; Touahni, R.; Messoussi. R.(2016). Automated Model Driven Testing Using AndroMDA and UML2 Testing Profile in Scrum Process. pp. 221–228.

O. Bertolino, A., Inverardi, P., & Muccini, H. (2001, July). An explorative journey from architectural tests definition down to code tests execution. In Proceedings of the 23rd International Conference on Software Engineering (pp. 211–220). IEEE Computer Society.

P. Javed, A. Z., Strooper, P. A., & Watson, G. N. (2007, May). Automated generation of test cases using model-driven architecture. In Automation of Software Test, 2007. AST'07. Second International Workshop on (pp. 3–3). IEEE.

Q. Lalchandani, J. T., & Mall, R. (2008, February). Regression testing based-on slicing of component-based software architectures. In Proceedings of the 1st India software engineering conference (pp. 67–76). ACM.

R. Lima, B., & Faria, J. P. (2015, July). An approach for automated scenario-based testing of distributed and heterogeneous systems. In Software Technologies (ICSOFT), 2015 10th International Joint Conference on (Vol. 1, pp. 1–10). IEEE.

S. Iyenghar, P., Westerkamp, C., Wuebbelmann, J., & Pulvermueller, E. (2010, September). An architecture for deploying model based testing in embedded systems. In Specification & Design Languages (FDL 2010), 2010 Forum on (pp. 1–6). IET.

T. Dan, H., & Hierons, R. M. (2011, March). Conformance testing from message sequence charts. In Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on (pp. 279–288). IEEE.

U. Havlice, Z., Szabóová, V., & Vízi, J. (2013, January). Critical knowledge representation for model-based testing of embedded systems. In Applied Machine Intelligence and Informatics (SAMI), 2013 IEEE 11th International Symposium on (pp. 169–174). IEEE.

V. Caliebe, P., Herpel, T., & German, R. (2012, April). Dependency-based test case selection and prioritization in embedded systems. In Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on (pp. 731–735). IEEE.

W. Carrozza, G., Faella, M., Fucci, F., Pietrantuono, R., & Russo, S. (2012, November). Integrating mdt in an industrial process in the air traffic control domain. In Software Reliability Engineering Workshops (ISSREW), 2012 IEEE 23rd International Symposium on (pp. 225–230). IEEE.

X. Wu, C. S., & Huang, C. H. (2013, May). The web services composition testing based on extended finite state machine and UML model. In Service Science and Innovation (ICSSI), 2013 Fifth International Conference on (pp. 215–222). IEEE.

Y. Schulz, S., Honkola, J., & Huima, A. (2007, March). Towards model-based testing with architecture models. In Engineering of Computer-Based Systems, 2007. ECBS'07. 14th Annual IEEE International Conference and Workshops on the (pp. 495–502). IEEE.

Z. Buchs, D., Pedro, L., & Lúcio, L. (2006). Formal test generation from UML models. Research Results of the DICS Program, 4028, 145–171.

AA. Lamancha, B. P., Reales, P., Polo, M., & Caivano, D. (2011, June). Model-Driven Test Code Generation. In International Conference on Evaluation of Novel Approaches to Software Engineering (pp. 155–168). Springer, Berlin, Heidelberg.

BB. Lima, B. M. C., & Faria, J. C. P. (2017, March). Towards Decentralized Conformance Checking in Model-Based Testing of Distributed Systems.

In Software Testing, Verification and Validation Workshops (ICSTW), 2017 IEEE International Conference on (pp. 356–365). IEEE.
CC. Wu, C. S., & Lee, Y. T. (2012, December). Automatic SaaS test cases generation based on SOA in the cloud service. In Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on (pp. 349–354). IEEE.
DD. Xu, D. (2011). A tool for automated test code generation from high-level Petri nets. Applications and Theory of Petri Nets, 308–317.
EE. Wehrmeister, M. A., & Berkenbrock, G. R. (2013, June). Automatic Execution of Test Cases on UML Models of Embedded Systems. In International Embedded Systems Symposium (pp. 39–48). Springer, Berlin, Heidelberg.

Appendix-C. Study quality assessment

	Quality of Reporting			Rigor			Credibility		Relevance		
Primary study	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Total
A	1	1	1	1	1	1	0	1	0.5	0.5	8
B	1	1	1	0,5	0,5	1	0	1	0,5	0,5	7
C	1	1	1	1	1	1	0	1	1	1	9
D	1	1	1	1	1	1	0	1	0,5	0,5	8
E	1	1	1	1	1	1	0	1	1	0,5	8,5
F	1	1	1	1	1	1	0	1	1	1	9
G	1	1	1	1	1	1	0	0,5	0,5	0,5	7,5
H	1	1	1	1	1	1	0	0,5	1	0,5	8
I	1	1	1	1	1	1	0	1	1	1	9
J	1	0,5	1	0,5	1	1	0	1	0,5	0,5	7
K	1	1	1	1	1	0,5	0	1	1	0,5	8
L	1	1	1	1	1	1	0	1	1	1	9
M	1	1	1	1	0,5	1	0	1	1	1	8,5
N	1	1	1	1	0,5	1	0	1	1	0,5	8
O	1	1	1	1	1	1	0	1	1	0,5	8,5
P	1	1	1	1	0,5	1	0	1	1	0,5	8
Q	1	1	1	1	1	1	0	1	1	1	9
R	1	1	1	1	0,5	1	0	1	1	1	8,5
S	1	1	1	1	1	1	0	1	0,5	1	8,5
T	1	1	1	1	0,5	1	0	1	1	1	8,5
U	1	1	1	1	1	1	0	1	0,5	1	8,5
V	1	1	1	1	1	1	0	1	1	1	9
W	1	1	1	1	1	1	0	1	1	1	9
X	1	1	1	1	0,5	1	0	0,5	1	0,5	7,5
Y	1	1	1	1	0,5	0,5	0	0,5	1	0,5	7
Z	1	1	1	1	1	1	0	1	1	1	9
AA	1	1	1	1	1	1	0	1	1	1	9
BB	1	1	1	0,5	1	1	0	1	1	0,5	8
CC	1	1	1	1	1	1	0	1	0,5	0,5	8
DD	1	1	1	1	1	1	0	1	1	0,5	8,5
EE	1	1	1	0,5	1	1	0	1	1	0,5	8

Appendix-D. Data extraction form

Study description	Extraction element	Contents
General Informatio-n		
1	ID	Unique id for the study
2	SLR 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	Addressed Concern	○ Code to Architecture Conformity ○ Internal Architectural Consistency Checking
11	Test Criteria	○ Coverage Criteria ○ Test Purpose Matching
12	Software Architecture Description Language	CHAM, Wright ADL, FSP Model, UML Diagrams, GoalML, AADL, Acme ADL, BPEL
14	Test Model	LTS , BG, Promela, LTL Formulae, Buchi Automata, IOLTS, Uppaal, HPrTNS, ECFG, eDeltaModels, Test Graph, U2TP Diagram
15	Test Case Execution	○ Automatic ○ Manual
16	Test Case Generation	○ Automatic ○ Manual
17	Test Oracle	○ Automatic ○ Manual
18	Assessment Approach	○ Case Study ○ Experiment ○ Small Example
19	Findings	
20	Constraints / Limitations	
Evaluation		
21	Personal note	The opinions of the reviewer about the study
22	Additional note	Publication details
23	Quality Assessment	Detailed quality scores

References

[1] [ANSI/IEEE 1059:1993], Software Verification and Validation Plan (ANSI/IEEE 1059), (2014).
[2] C. Baier, J.P. Katoen, K.G. Larsen, Principles of Model Checking, MIT Press, 2008.
[3] E. Clarke, O. Grumberg, D. Peled., Model Checking, MIT Press, 2000.
[4] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, J. Stafford, Documenting Software Architectures: Views and Beyond, second ed., Addison-Wesley, 2010.
[5] E. Demirli, B. Tekinerdogan, Software language engineering of architectural viewpoints, Proc. of the 5th European Conference on Software Architecture (ECSA 2011), LNCS 6903, 2011, pp. 336–343.
[6] M. Felderer, P. Zech, R. Breu, M. Büchler, A. Pretschne, Model-based security testing: a taxonomy and systematic classification, J. Software (2015), http://dx.doi.org/10.1002/stvr.1580.
[7] M. Fowler, Domain-Specific Languages, Addison-Wesley Professional, 2011.
[8] V. Garousi, MikaV. Mäntylä, When and what to automate in software testing? A multi-vocal literature review, Inf. Software Technol. 76 (2016) 92–117, http://dx.doi.org/10.1016/j.infsof.2016.04.015.
[9] H.G. Gurbuz, B. Tekinerdogan, Model-based testing for software safety: a systematic literature review, Software Qual. J. (2017), http://dx.doi.org/10.1007/s11219-017-9386-2.
[10] A. Hartman, M. Katara, S. Olvovsky, Choosing a test modeling language: a survey. doi:10.1007/978-3-540-70889-6_16, 2006.
[11] ISO/IEC 42010:2007], Recommended Practice for Architectural Description of Software-Intensive Systems (ISO/IEC 42010), (2011).
[12] B. Kitchenham, S. Charters, Guidelines for performing systematic literature reviews in software engineering, in: E.T. Report (Ed.), Engineering, 2(EBSE 2007-001), 1051 2007, http://dx.doi.org/10.1145/1134285.1134500 2007.
[13] B. Kitchenham, D. Budgen, P. Brereton, Evidence-Based Software Engineering and Systematic Reviews, CRC Press Taylor & Francis Group, 2016.
[14] M.A. Mäkinen, Model Based Approach to Software Testing, Thesis Helsinki University of Technology Department of Computer Science and Engineering Software Business and Engineering Institute, 2007.
[15] G.J. Myers, C. Sandler, T. Badgett, The Art of Software Testing, John Wiley & Sons, Inc, Hoboken, New Jersey, 2012.
[16] H. Muccini, A. Bertolino, P. Inverardi, Using software architecture for code testing, IEEE Trans. Software Eng. 29 (March (3)) (2003) 160–171.
[17] S.J. Mellor, K. Scott, A. Uhl, D. Weise, MDA Distilled: Principle of Model Driven Architecture, Addison Wesley, Reading, 2004.
[18] I. Mistrík, R.M Soley, N. Ali, J. Grundy, B. Tekinerdogan, Software Quality assurance: in Large Scale and Complex Software-Intensive Systems, Morgan Kaufmann, 2015.
[19] G. Murphy, D. Notkin, K. Sullivan, Software reflexion models: bridging the gap between design and implementation, IEEE Trans. Software Eng. 27 (4) (2001) 364–380.
[20] A.C.D. Neto, R. Subramanyan, M. Vieira, G.H. Travassos, A survey on model-based testing approaches: a systematic review, Proceedings of the 2007 ACM Symposium on Applied Computing, 2007.
[21] Cu.D. Nguyen, A. Marchetto, P. Tonella, Combining model-based and combinatorial testing for effective test case generation, Proceedings of the 2012 International Symposium on Software Testing and Analysis (ISSTA 2012). New York,USA, ACM, 2012, pp. 100–110 DOI = http://dx.doi.org/10.1145/2338965.2336765.
[22] D.M. Rafi, K.R.K. Moses, K. Petersen, M.V. Mantyla, Benefits and limitations of automated software testing: Systematic literature review and practitioner survey, Automation of Software Test (AST), 2012 7th International Workshop on, 42 2012, p. 36 vol., no.2–3June.
[23] N.M. Souza, D. Dias, L.B. Ruas de Oliveira, C. Aparecida Lana, E. Yumi Nakagawa, J.C. Maldonado, Exploring together software architecture and software testing: a systematic mapping, International Conference of the Chilean Computer Science Society - SCCC2016, Valparaíso, Chile, 35 2016 October.
[24] B. Tekinerdogan, N. Ali, J. Grundy, I. Mistrik, R. Soley, Quality concerns in large scale and complex software-intensive systems, in: I. Mistrik, R. Soley, N. Ali, J. Grundy, B. Tekinerdogan (Eds.), Software Quality Assurance in Large Scale and Complex Software-Intensive Systems, Elsevier, 2015, pp. 1–17.
[25] B. Tekinerdogan, software architecture, in: T. Gonzalez, J.L. Díaz-Herrera (Eds.), Second Edition, Computer Science Handbook, vol. I, Computer Science and Software Engineering, Taylor and Francis, 2014.
[26] B. Tekinerdogan, Architectural drift analysis using design structure reflexion matrices. In: I. Mistrik, R. Soley, N. Ali, J. Grundy, B. Tekinerdogan. Sofware Quality Assurance in Large Scale and Complex Software-Intensive Systems Elservier, - p. 221–236.
[27] M. Utting, A. Pretschner, B. Legeard, A taxonomy of model-based testing approaches, Software Test. Verif. Reliab. (2011).
[28] M. Utting, B. Legeard, Practical Model-Based Testing: A Tools Approach, Morgan Kaufmann, 2007.
[29] P. Zhang, H. Muccini, B. Li, A classification and comparison of model checking software architecture techniques, Journal of Systems and Software 83 (5) (2010) 723–744.