Model-Driven Architecture Based Testing: A Systematic Literature Review

Uzun, B., & Tekinerdogan, B.

This is a "Post-Print" accepted manuscript, which has been Published in "Information and Software Technology"

This version is distributed under a non-commercial no derivatives Creative Commons

([CC-BY-NC-ND](#)) user license, which permits use, distribution, and reproduction in any medium, provided the original work is properly cited and not used for commercial purposes. Further, the restriction applies that if you remix, transform, or build upon the material, you may not distribute the modified material.

Please cite this publication as follows:

Uzun, B., & Tekinerdogan, B. (2018). Model-Driven Architecture Based Testing: A Systematic Literature Review. Information and Software Technology, 102, 30-48. <https://doi.org/10.1016/j.infsof.2018.05.004>

You can download the published version at:

<https://doi.org/10.1016/j.infsof.2018.05.004>

Model-Driven Architecture Based Testing: A Systematic Literature Review

Burak Uzun¹, Bedir Tekinerdogan²

¹Information Technology Group, Wageningen University, Wageningen, The Netherlands
burak.uzun@wur.nl

²Information Technology Group, Wageningen University, Wageningen, The Netherlands
bedir.tekinerdogan@wur.nl

Context: Model-driven architecture based testing (MDABT) adopts architectural models of a system under test and/or its environment to derive test artefacts. In the literature, different MDABT approaches have been provided adopted together with the corresponding lessons results and lessons learned.

Objective: The overall objective of this paper is to provide a systematic literature review (SLR) to identify, analyze and describe the state of the art advances in MDABT.

Method: The systematic review is conducted by a multi-phase study selection process using the published literature in major software engineering journals and conference proceedings.

Results: We reviewed 165 papers that are discovered using a well-planned review protocol, and 14 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.

Keywords: model-based testing, software architecture, systematic review

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]. 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. As a result, selecting set of test

cases which can detect possible flaws of the system is the key challenge in software testing. *Model based testing* (MBT) addresses this challenge by automating the generation and execution of test cases using models based on system requirements and behavior [13]. MBT relies on models to automate the generation of the test cases and their execution. 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. A recent trend in MBT is to adopt software architecture models to provide automated support for the test process 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.

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 well as the variations, and the guidelines and lessons learned are actually scattered over different studies in the literature. To provide a clear overview of MDABT we adopt a systematic literature review (SLR) approach based on Kitchenham's guidelines [12].

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. Section 3 discusses the overall protocol of the adopted SLR. Section 4 presents the results of the adopted SLR protocols. Section 5 presents discussion over the SLR. 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. According to Mellor et al. [14] an executable model is a model that has everything required to produce the desired functionality of a single domain. Executable models are more precise than sketches or blueprints, 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. [14]. 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. In model-driven development, models are usually organized in a four-layered architecture. The top (M3) level in this model is the so called meta-metamodel, and defines the basic concepts from which specific meta-models are created at the meta (M2) level. Normal user models are regarded as residing at the M1 level, whereas real world concepts reside at level M0.

Figure 1. Process of model based testing

Model-based testing builds on model-driven development in which models are used to automate the testing process. According to Utting et al. [22] 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 to perform model based testing such as easy test maintenance, automated test design and enhancing test quality. In Figure 1 (adapted from [22]) 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

Architectural drivers define the concerns of the stakeholders which shape the architecture [3]. 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. 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. 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 [3] 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 provide a systematic literature review (SLR) to identify, analyze and describe the state of the art advances in MDABT. For this we will apply a systematic literature review (SLR) or systematic review for short 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]. 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 and Charters [12]. 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 protocol in the subsequent sub-sections.

3.1. Review Protocol

The adopted review protocol is shown in Figure 2. First, we specified our research questions based on the objectives of this systematic review. Then 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 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 was defined the data extraction strategy was developed which defines how the information required from each study is obtained. For this we developed a data extraction form that was defined after a pilot study. In the final step the data synthesis process takes place in which we present the extracted data and associated results.

Figure 2. The adopted review protocol

3.2. Research Questions

The fundamental step of the SLR is identification of specific and valid research questions. The selected studies after applying review protocol must answer all research questions defined. Research question must be valid in order to be answered by each study in the domain.

We have identified the following research question which should be answered by the selected set of studies. In this study, we are interested in conducting the following research questions which are:

RQ.1: What are the addressed concerns for applying model-driven software architecture based testing?

RQ.2: What are the proposed solutions in architecture-based testing?

RQ.3: What are the existing research directions within architecture-based testing?

3.3. Search Strategy

One of the main motivation of SLR is to identify primary studies answering the specified research questions utilizing well-defined search strategy. Our search scope includes two attributes which are publication date range and publication platforms. We have selected the range of dates between period of 2000 and April 2017 for our publication date range attribute. We have selected start date as 2000 because first paper having strong foundations on the topic with a proper case study explaining overall method is published at this date. We searched in publication platforms of IEEE Xplore, ACM Digital Library, Wiley, Science Direct, Springer, and ISI Web of Knowledge. To search a database we used two search methods, automatic search and manual search. Automatic 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. Since the selected databases include thousands of published papers, manual search becomes too time-consuming. For this reason, we conducted automated search by using search string.

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	55	26	4
ACM Digital Library	8	8	2
Wiley Interscience	31	10	0
Science Direct	57	15	5
Springer	4	3	2
ISI Web of Knowledge	10	6	1
Total	165	68	14

We have identified a search string for each publication platform listed in our search scope for retrieving as much possibly related studies we can. 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 platform search language. The created queries are 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 Search Strings. Table 1 presents the results of the overall search process. The first column provides the searched venues. In the second column of the table, 165 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 explained in the following sub-

section. In the last step of the process 14 studies have been identified as primary studies to be used for detailed data analysis and synthesis.

3.4. Study Selection Criteria

Search query strings provide large range of primary studies over the domain so that we will not miss any related studies. Study selection criteria is defined so that studies that are not directly related to our domain are excluded. Defined criteria are manually applied on the set of selected studies. In the following part, we provide exclusion 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 don't validate the proposed study
- EC 8: Papers that provide only a general summary without a clear contribution

3.5. Study Quality Assessment

We have defined a method for study quality assessment method for giving further detailed exclusion or inclusion criteria. The importance or validness of the study is examined by quality assessment method we provide. Our defined method is based on quality attributes which are checklist of properties that each study assessed by [1]. The checklist created by taking in account the properties that could bias study results. Defined quality assessment method maintain the summary quality checklist for quantitative studies and qualitative studies which is proposed on [1].

Table 2. Adopted Quality Checklist

No	Question
Q1	Are the aims of the study is clearly stated?
Q2	Are the scope and context of the study clearly defined?
Q3	Is the proposed solution clearly explained and validated by an empirical study?
Q4	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?

Table 2 provides our quality checklist for quality assessment method. Our main motivation in adopting quality checklist is to assess a study by an overall quality score. 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 14 selected primary studies for answering each research question. Furthermore, a data extraction form is designed for retrieving all the information to answer 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 data that enables us to result the foundation of both perspectives. 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 for both perspectives of the selected studies.

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 14 selected primary studies. Figure 3 shows the publication year distribution of the selected primary studies.

Figure 3: Year-wise distribution of primary studies

Figure 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.

Table 3: Distribution of the studies over Publication Channel

Publication Channel	Publication Source	Type	Number of Studies
Information and Software Technology	ScienceDirect	Article	1
Formal Methods for Software Architectures	Springer	Chapter	1
Fundamental Approaches to Software Engineering	Springer	Chapter	1
Applying Formal Methods: Testing, Performance, and M/E-Commerce	Springer	Article	1
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
Proceeding ROSATEA '06 Proceedings of the ISSTA 2006 workshop on Role of software architecture for testing and analysis	ACM	Article	1
Electronic Notes in Theoretical Computer Science	ScienceDirect	Article	1
Information Technology: New Generations (ITNG), 2010 Seventh International Conference	IEEE	Conference	1
Information and Software Technology	ScienceDirect	Chapter	1
Journal of Systems and Software Volume 91	ScienceDirect	Article	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

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 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 short example. From the table, we can conclude that case study research method is the most frequently adopted approach in the selected studies.

Table 4. Distribution of studies over Research Method

Research Method	Studies	Number	Percent
Case Study	A,C,D,E,F,H,K,M	8	57
Experiment	B,L	2	14
Short Example	G,J, N	3	21
None	I	1	8

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. The values for the quality attributes are derived from the quality checklist which was given in the previous section. The first three question of the quality checklist relate to the reporting quality attribute, while the last two questions are used for relevance quality. 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. In Appendix-C the result of the quality checklist is presented.

Figure 4 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 1 of the primary studies is close to the highest score.

Figure 4. Quality of reporting of the primary studies

Rigor quality of the study refers to the trustiness of findings of the study. Figure 5 shows that 65% of the studies have the highest score in terms of rigor quality. Moreover, 28 % of the studies have been assessed as very good. However, 7% of the studies ranked as good rigor quality.

Figure 5. Rigor quality of the primary studies

Another quality measure is the relevance quality of the primary studies. Figure 6 shows relevance quality scores calculated from ninth and tenth question of the quality checklist. It can be seen that 14% of the studies is directly relevant to MDABT. The rest of the studies are relevant to MDABT.

Figure 6. Relevance quality of the primary studies

The credibility quality of the studies is calculated by using the seventh and eight question. Figure 7 presents the credibility quality score and distribution of the studies. It can be seen that 85% of the studies calculated as 1 point. Remaining 15% 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.

Figure 7. Credibility of evidence of the primary studies

The summary of the overall methodological quality scores of selected primary studies is given in Figure 8. Total quality is calculated by adding up reporting, relevance, rigor and credibility qualities. We consider 8.5-9 as high quality, 7.5-8 very good quality and 7 as good quality. It can be seen that 42% studies have high overall quality. Moreover, 42% of the studies have very good overall quality and 16% of the studies have good overall quality.

Figure 8. Overall quality of the primary studies

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 essence two different 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 are shown in Figure 9. The functional concerns are functional requirements that are tested on architecture or code level. For instance, architecture level functional requirement can be graph representation of architecture cannot have tangling node whereas at the code level this can be the validation of the functional requirement.

Figure 9. Addressed concern distribution of primary studies

Table 5 Addressed Concerns and Studies

Addressed Concerns	Studies
Code to architecture conformity	A,C,D,E,G,H,I,K,L,M, N
Internal Architectural Consistency Checking	B,F,J

In study A, the authors explain how to derive test plans from software architecture (SA) specifications. The testing is performed for conforming implemented system with the specified SA. The main motivation of this paper is to use SA specifications for integration testing of the implemented system.

Study C, is the continuation of the study A the concerns and the motivation is the same. In this study authors explain how they fill the gap between the abstraction of SA and implementation using architectural and code level sequence diagrams.

In study B, authors define test criterion to use in SA based testing. The testing is done at the architecture design level to test functional properties of SA. The main motivation of this paper is to define formal testing criteria based on architectural relations.

In study D, authors explain how they provide systematic way to perform the refinement step. In this study, specific architectural style is used which has mapping between SA and code based test cases to delivery completely systematic SA based approach. The testing is performed for code to architecture conformance. The main motivation is to exploit architectural style information to create mapping between component in SA and units in implementation.

In study E, authors use both model checking and SA based testing approaches. The addressed concern of this study is code to architecture conformance. The main motivation is to select architecture level test cases using the output generated from model checking stage.

In study F, authors perform SA based testing at architectural level for testing specifications against functional properties of SA. The main motivation of this paper is to validate architectural units using object oriented models.

In study G, the addressed concern is 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.

In study H, authors used their previous works on SA based testing for regression testing of software system. The addressed concern of code to architecture code conformity is regressively tested. The main motivation of this paper is to use SA based testing for regression testing of systems.

In study I, both model checking of SA and SA based testing approaches are experimented. The addressed concern of the study is code to architecture conformity. The main motivation of this paper is to use Architecture Analysis and Design Language (AADL) specifications in verification of software system.

Study J, addresses the concern of functional properties of SA. The testing is performed at architectural level. The main motivation of this paper is to use SA in model based testing to detect defects earlier in software lifecycle.

In study K, service oriented applications (SOA) are tested which is said to be more challenging than monolithic applications. The main motivation of this paper is to use SA based testing to solve observability and controllability problems that are created by message exchanges between the services that are hidden behind the front service of the system. The addressed concern of this study is code to architecture conformity.

In study L, authors use SA based testing approaches to variant-rich software systems. The main motivation of the study is to address the challenge of ensuring correctness of component implementations and interactions with any system variant. The addressed concern of this study is code to architecture conformity.

In study M, authors utilize architecture based testing using behavioral UML models for generating behavior driven tests for cucumber tool. The proposed approach and tool addresses the concern of conforming given architecture to implementation.

In study N, authors suggest automated model driven testing approach using UML sequence diagrams with motivation of ensuring the correctness of implementation with respect to code.

RQ2: What are the proposed solutions in 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 Figure 10. 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.

Figure 10. Generic Process for Architecture-Based Testing

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.

- *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 Figure 10 is based on a general model-based testing approach. In this study we do not focus in models in general but focus on architectural models for supporting model-based testing. In principle, different ABT approaches can be derived from the same 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.

Study A

In Figure 11 the process model of this study is shown. 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. Test model is generated using CHAM specifications and coverage criteria which is a transition graph called label transition system (LTS). From this graph, abstract labeled 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.

Figure 11. Process Model Applied in Study A

Study B

Figure 12 shows the process model of this study. This study presents testing at the architecture level for conformance of SA properties which are pointed as test criterion they define. In this case, 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.

Figure 12. Process Model Applied in Study B

Study C

Figure 13 shows the process model of this study. This study presents the ongoing work of previous authors in A by replacing their SA specification of CHAM model with an FSP (Finite State Process) model. The provided reason to use FSP instead of CHAM model is that FSP algebra is easier to map to LTS graph. In this study, the authors explain the work of the software architect to generate test cases and execute them manually. This study presents the use of UML Stereotyped Sequence Diagrams 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.

Figure 13. Process Model Applied in Study C

Study D

Figure 14 shows the process model of this study. This study presents the use of architecture specification for mapping the SA model to implementation. Different from the study C the authors have now added automatic test case generation and analysis reporting.

Figure 14. Process Model Applied in Study D

Study E

Figure 15 shows the process model for study E. This study uses the same methodology explained in study D. Moreover, they present model checking for software architecture using SPIN and 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, LTL Formulae and Buchi Automata. The test cases that were ALTS paths are chosen as OK and NOK results that are obtained from model checking stage. OK and NOK results stands for successful and unsuccessful verifications obtained after applying model checking on SA. The results are used by test generator engine to create test cases automatically at the SA level. The test execution and test analysis is handled manually.

Figure 15. Process Model Applied in Study E

Study F

Figure 16 shows the process model of this study. This 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 LOTOS (Language of Temporal Ordering Specifications). Basic LOTOS model is combined with test purposes to generate IOLTS model where test cases are generated using TGV tool. The following figure shows how this study maps to our process model.

Figure 16. Process Model Applied in Study F

Study G

Figure 17 shows the process model of this study. Hereby, system behavior is implemented by system level plan where system level plan consists of sequence of goals each describing the interactions between the components. Moreover, component level plan shows the sequence that component must achieve in that scenario. Together system level plan and component level plan provide specifications for SA. Test criteria is the scenario that is executed on the system described the interactions in system level plan and component level plan. System level plan and component level plan is used by JESS tool to annotate the implementation code. Then by the pre-compiler the annotations are transformed into codes. 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 and if all the component level plans are matched system level plan matches. If the system level plan matches in the executed program then the program passes the test.

Figure 17. Process Model Applied in Study G

Study H

In this study, same methodology in study D is used and same process model in Figure 14 applies.

Study I

This study presents the use of Architecture Analysis and Design Language (AADL) to specify SA which is 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.

Figure 18. Process Model Applied in Study I

Study J

This study presents the use of HPrTNs (a Petri Net) 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.

Figure 19. Process Model Applied in Study J

Study K

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 BPEL (Business Process Execution Language). In the SA specifications architecture topology and message exchange knowledge are included. The extended control flow graph (ECFG) test model is used to 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.

Figure 20. Process Model Applied in Study K

Study M

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. UML message sequence diagrams are used for representing component interactions of SA. UML component diagrams are used for representing topological view 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. In this test model, different views are obtained using signals as reducing function for state explosion problem. 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. Delta SM tool is used for constructing component state machine. DeltARX is used for constructing class diagram model. IBM rational rhapsody is used for test case generation. Test execution and analysis is automatically handled.

Figure 21. Process Model Applied in Study L

Study M

In this study, a tool for generating behavior driven tests using behavioral UML diagrams based on EMF 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 are executed automatically via another tool called Cucumber. Testers then writes mappings for each generated Cucumber scenarios and execute tests. The proposed tool is compatible with continuous integration environments. In the Figure 22 the process model adaptation of the study is presented.

Figure 22 Process Model Applied in Study M

Study N

In this study, authors propose an approach 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. Later test models are transformed into concrete test cases using AndroMDA tool in which mappings are given in the format of VSL files for generation. 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. In Figure 23 process model adaptation of proposed study is shown.

Figure 23. Process Model Applied in Study N

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 Figure 10. On the 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 to understand each of these processes and compare them. Obviously, the SA specification appeared to be an important part of our process model and different approaches applied different modeling approaches. Figure 24 shows the distribution of the various SA specifications over the approaches from the primary studies. As we can observe from the figure, UML and FSP model appeared to be the most frequent used specification approaches.

Based on the reference process, the second differing aspect in the approaches is the adopted test models from which test cases are generated. Figure 25 shows the test model distribution over the various approaches. 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. One of the studies where SA is specified by GoalML domain specific language did not use any test models. However, the tested properties are annotated in the implementation itself and during the execution of the system the assertions are emitted.

Thirdly, the test criteria is an important property for refining the test model from SA specifications. Figure 26 shows the distribution of test criteria over the different approaches. As the test models are based on graphs most of the studies used coverage criteria when generating test cases. Three of the studies used test purposes where test purposes are properties

that are refined to be tested. For example, in approaches where SA is specified using goals the test criteria is to match the component and system level plans.

Fourthly, test case generation type is another important property in model driven SA based testing. In Figure 27 test case generation type distributions over the studies can be seen. Most of the studies 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. Moreover, study that applied successfully using automated test case generation at SA level performed manual when testing at code level without using architecture style. However, using the architecture style properties this stage can be automated as done in study D.

Test analysis is the last part of our model where the results of the tests that are executed are analyzed. In Figure 28 distribution of test analysis types over studies can be seen. Most of the studies used automated analysis where the result of failing or passing is automatically determined.

Table 6 summarizes the results as described above.

Figure 24. The Identified Architecture Description Approaches and the Frequency Over The Different Studies

Figure 25. The Identified Test Model Representations

Figure 26. The Identified Test Criteria

Figure 27. The Identified Test Case Generation Type

Figure 28. The Identified Test Analysis Type

Table 6. Overview of the identified approaches

Study	Architecture Model	Test Model	Test Criteria	Test Case Generation	Test Analysis
A	CHAM	LTS	Coverage Criteria	Manual	Manual
B	Wright ADL	BG	Coverage Criteria	Manual	Automated
C	FSP	LTS	Coverage Criteria	Manual	Manual
D	FSP	LTS	Coverage Criteria	Manual	Automated
E	UML	Promela, LTL Formulae, Buchi Automata	Test Purpose Matching	Manual	Manual
F	UML	IOLTS	Test Purpose Matching	Automated	Automated
G	GoalML	NA	Test Purpose Matching	Automated	Automated
H	FSP	LTS	Coverage Criteria	Automated	Automated
I	AADL	Uppaal	Coverage Criteria	Automated	Automated
J	ACME ADL	HPrTNS	Coverage Criteria	Automated	Automated
K	BPEL	ECFG	Coverage Criteria	Automated	Automated
L	UML	eDeltaModels	Coverage Criteria	Automated	Automated
M	UML	Test Graph	Coverage Criteria	Automated	Automated
N	UML	U2TP Sequence Diagram	Coverage Criteria	Automated	Automated

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

After we have characterized each model-driven architecture-based testing approach we will now discuss the existing research directions. Based on the identified primary studies we summarize the research directions as follows:

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. 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 studies used 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

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. There is a need for filling the gap between the SA and implemented system. One of the studies have proposed to use architecture styles for filling the gap between the SA level and implementation level. Still there is a need to exploit architecture definitions at implementation level to generate test cases.

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. Although the adoption of model-driven approaches 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. There are though still many challenges to find more effective criteria while generating test cases.

Architecture views

As discussed in the background, architecture needs to be modeled using multiple architecture views to represent the various stakeholder concerns. The approaches that we have considered however 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. Yet, existing architecture viewpoints aim to provide a different perspective on the system based on different stakeholders. Based on this, different viewpoints are distinguished such module views, component-connector views and allocation views. Integrating architecture views, as such, will be a necessary enhancement to the existing architecture-based testing approaches.

Need for automating the process by tools and cost-benefit analysis

When using model driven techniques in some context it is important to automate processes and assessing methods performance by some metrics defined from the context. Interestingly, the majority of the studies still rely on manual intervention at one or more of the activities. The automation of the several stages have been studied by different authors in their works. However, none of the work above have the complete autonomous system for MDABT based on our reference process model. 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 define benefits of the MDABT according to the cost of the methodology presented. Some authors have stated to work on this issue in their future works.

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 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 need for applying MDABT on more complex industry cases to justify the effectiveness of these approaches.

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 were 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. Conclusion

Model based testing (MBT) aims to automate the generation and execution of test cases using models based on system requirements and behavior. So far, various different models have been adopted to derive the test cases. 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 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 165 papers that are discovered using a well-planned review protocol, and 14 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. The analysis of the primary studies showed that all the presented approaches could be considered as instantiations of the reference process model. 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. Although each approach appears to be interesting we can have two basic observations in this context. Firstly, the current approaches also have some obstacles which need to be solved. We have discussed these obstacles including 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 and cost-benefit analysis, and the need for applying MDABT on complex real systems to provide evidence of the approaches with respect to conventional approaches. The second observation that we can derive is the fact 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 14 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. 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.

References

- [1]. [ANSI/IEEE 1059:1993] Software Verification and Validation Plan (ANSI/IEEE 1059), May 2014.
- [2]. D. Atkins, D. Best, P.A. Briss, M. Eccles, Y. Falck-Ytter, S. Flottorp, G.H. Guyatt, R.T. Harbour, M.C. Haugh, D. Henry, S. Hill, R. Jaeschke, G. Leng, A. Liberati, N. Magrini, J. Mason, P. Middleton, J. Mrukowicz, D. O'Connell, A.D. Oxman, B. Phillips, H.J. Schünemann, T.T.-T. Edejer, H. Varonen, G.E. Vist, J.W. Williams, S. Zaza, Grading quality of evidence and strength of recommendations, *BMJ (British Medical Journal)* 328 (2004) 1490.
- [3]. Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Merson, P., Nord, R., & Stafford, J. Documenting Software Architectures: Views and Beyond. Second Edition. Addison-Wesley, 2010.
- [4]. Dybå, T., Dingsøyr, T. Strength of Evidence in Systematic Reviews in Software Engineering, in: Proceedings of the 2nd ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'08), ACM, 2008, pp. 178–187.
- [5]. Fraser, G., Wotawa, F. and Ammann, P. 2009. Testing with model checkers: a survey. *Software Testing, Verification and Reliability*, 19 (3), pp. 215–261.
- [6]. Hartman, A., Katara, M., & Olvovsky, S. (2006). Choosing a Test Modeling Language: A Survey. doi:10.1007/978-3-540-70889-6_16

- [7]. Hessel A, Pettersson P. Cover—A Real-time Test Case Generation Tool. Accepted for the 3rd Workshop on Model-Based Testing 2007 (MBT07).
- [8]. Holzmann, G. 2013. Spin - Formal Verification. [online] Available at: <http://spin-root.com/spin/whatispin.html>
- [9]. Hunn, M., Hungar, H. (2010). UML For Software Safety and Certification. In: Giese, H. eds. *Model-based Engineering of Embedded Real-time Systems*. Springer (2010).
- [10]. IEEE Standard for Software Test Documentation, IEEE Std 829-1998
- [11]. ISO/IEC 42010:2007] Recommended practice for architectural description of software-intensive systems (ISO/IEC 42010), 2011.
- [12]. Kitchenham, B., Charters, S. Guidelines for performing Systematic Literature Reviews in Software Engineering. (E. T. Report, Ed.) Engineering, 2(EBSE 2007-001), 1051, 2007 doi:10.1145/1134285.1134500
- [13]. Mäkinen, M.A., Model Based Approach to Software Testing. Thesis, Helsinki University of Technology Department of Computer Science and Engineering Software Business and Engineering Institute, 2007.
- [14]. S.J. Mellor, K. Scott, A. Uhl, D. Weise. MDA Distilled: Principle of Model Driven Architecture, Addison Wesley, Reading, 2004.
- [15]. Moura, L. 2013. Symbolic Analysis Laboratory. [online] Available at: <http://sal.csl.sri.com/>
- [16]. Nusmv.fbk.eu. 2013. NuSMV home page. [online] Available at: <http://nusmv.fbk.eu/>
- [17]. Object Management Group: UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE), Beta 2 (2008)
- [18]. Prowell, S. 2003. JUMBL: A tool for model-based statistical testing, in HICSS '03: Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 9. IEEE Computer Society, 2003, p. 337.3.
- [19]. Rafi, D.M.; Moses, K.R.K.; Petersen, K.; Mantyla, M.V., Benefits and limitations of automated software testing: Systematic literature review and practitioner survey, Automation of Software Test (AST), 2012 7th International Workshop on, vol., no., pp.36, 42, 2 -3, June 2012 doi: 10.1109/IWAST.2012.6228988
- [20]. Testingeducation.org. 2013. Examples of Test Oracles. [online] Available at: <http://www.testingeducation.org/k04/OracleExamples.htm>
- [21]. Thomas, F., Delatour, J., Terrier, F., Gérard, S.: Towards a framework for explicit platform-based transformations. In: 11th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC), pp. 211–218. Los Alamitos (2008)
- [22]. Utting, M., Pretschner, A., Legeard, B. A taxonomy of model-based testing approaches. Software Testing, Verification and Reliability, 2011
- [23]. Utting M., Position Paper: Model-Based Testing, Verified Software: Theories, Tools, Experiments. ETH Zürich, IFIP WG
- [24]. Zoughbi, G., Briand, L., Labiche, Y.: A UML profile for developing airworthiness-compliant (RTCA-DO-178B) safety-critical systems. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F. (eds.) MODELS 2007. LNCS, vol. 4735, pp. 574–588. Springer, Heidelberg (2007)

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	((Title:model based testing" OR Title:model based software testing" OR Title:model-based testing" OR Title:model-based software testing" OR Title:model driven testing" OR Title:model driven software testing" OR Title:model-driven testing" OR Title:model-driven software testing" OR Title:model based test" OR Title:model based software test" OR Title:model-based test" OR Title:model-based software test" OR Title:model driven test" OR Title:model driven software test" OR Title:model-driven test" OR Title:model-driven software test") AND (Title: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-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")
Wiley Interscience	("model based testing" OR "model based software testing" OR "model-based testing" OR "model-based software testing" OR "model driven testing" OR "model driven software testing" OR "model-driven testing" OR "model-driven software testing" OR "model based test" OR "model based software test" OR "model-based test" OR "model-based software test" OR "model driven test" OR "model driven software test" OR "model-driven test" OR "model-driven software test") AND ("architecture based testing" OR "architecture based software testing" OR "architecture-based testing" OR "architecture-based software testing" OR "architecture driven testing" OR "architecture driven software testing" OR "architecture-driven testing" OR "architecture-driven software testing" OR "architecture based test" OR "architecture based software test" OR "architecture driven test" OR "architecture driven software test" OR "architecture-based test" OR "architecture-based software test" OR "architecture driven test" OR "architecture-driven test" OR "architecture- driven software test")

Science Direct	<p>((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)))</p> <p>OR (</p> <p>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)))</p> <p>OR</p> <p>((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)))</p> <p>OR (</p> <p>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)))</p>
Springer	<p>("model based testing" OR "model based software testing" OR "model-based testing" OR "model-based software testing" OR "model driven testing" OR "model driven software testing" OR "model-driven testing" OR "model-driven software testing" OR "model based test" OR "model based software test" OR "model-based test" OR "model-based software test" OR "model driven test" OR "model driven software test" OR "model-driven test" OR "model-driven software test") AND ("architecture based testing" OR "architecture based software testing" OR "architecture-based testing" OR "architecture-based software testing" OR "architecture driven testing" OR "architecture driven software testing" OR "architecture-driven testing" OR "architecture-driven software testing" OR "architecture based test" OR "architecture based software test" OR "architecture-based test" OR "architecture-based software test" OR "architecture driven test" OR "architecture driven software test" OR "architecture-driven test" OR "architecture-driven software test")</p>
ISI Web of Knowledge	<p>((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"))</p> <p>OR (</p> <p>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"))</p>

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.; Bertolino, 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.

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

APPENDIX-D DATA EXTRACTION FORM

Study de- scription	Extraction element	Contents
General Information		
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