Adaptation of Software Testability Concept for Test Suite Generation
- A Systematic Review

Prakash Malla
Bhupendra Gurung

School of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona
Sweden
This thesis is submitted to the School of Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:
Authors:
Prakash Malla  
Personal Number: 771021-3591  
Address: Lindblomsvagen 96, LGH 1105, 37233, Ronneby, Sweden  
E-mail: pkmalla@gmail.com  

Bhupendra Gurung  
Personal Number: 820825-9799  
Address: Lindblomsvagen 96, LGH 1105, 37233, Ronneby, Sweden  
E-mail: bhupendra_grg@hotmail.com  

University Advisor:  
Kennet Henningsson  
School of Computing,  
Blekinge Institute of Technology,  
Karlskrona  
E-mail: Kennet.Henningsson@bth.se  

School of Computing  
Blekinge Institute of Technology  
SE-371 79 Karlskrona  
Sweden  
Internet : www.bth.se/com  
Phone : +46 455 38 50 00  
Fax : +46 455 38 50 57
ABSTRACT

Context: Software testability, which is the degree to which a software artifact facilitates process of testing, is not only the indication of the test process effectiveness but also gives the new perspective on code development. Since more than fifty percent of total software development costs is related to testing process activities, Software testability has always been the improving area in software domain so that we can make the software development process effective with respect to test cases writing and fault detection process.

Objectives: The research though this thesis will have the objective of proposing a conceptual framework considering the testability issues for the simpler test suite generation and facilitating the concerned persons with better effectiveness of testing. We investigate the testability factors and testability metrics basically with the help of the systematic literature review and the proposed framework’s feasibility is evaluated with case study.

Methods: Initially, we conduct the literature review to get broad knowledge on this domain as well for the key documents. Then study starts with the systematic literature review process guided by the review protocol to collect the testability factors and measurements. The framework is validated with the case study. The research documents are included from highly trusted e-database including Compendex, Inspec, IEEE Xplore, ACM Digital Library, Springer Link and Scopus. Altogether 36 primary documents are included for the study and results are extracted.

Results: From the results of systematic literature review, Software testability factors and associated measurements are found and the construction of framework for simple test generation as guidelines evaluate with case study. To make the test suite generation simpler, we propped a framework based on the FTA concepts and breakdown of high level testability factors to its simpler form of measurable level.

Conclusions: Numbers of different software testability factors are presented in different researches in different perspectives. We collect important testability factors and associated measurement methods and we concluded the effect of testability in simpler test suite generation with the help of framework evaluated by case study.

Keywords: software testability, test suite generation, testability framework, testability factors, testability measurements.
ACKNOWLEDGEMENT

We would like to express our sincere thanks to the people who has facilitated us in completing our thesis. First of all, we would like to thank our supervisor Kennet Henningsson for his invaluable guidance. We have found him always encouraging us and supporting for the necessary supervision and help to find the tools required for the thesis work. We would like to thank to the library staffs who have helped us for database management, searching strategies and managing the references. At the end we would like to thank for the participant of case study.
CONTENTS

ABSTRACT ........................................................................................................................................... I

ACKNOWLEDGEMENT ......................................................................................................................... II

CONTENTS ............................................................................................................................................... III

LIST OF TABLES ..................................................................................................................................... V

LIST OF FIGURES ................................................................................................................................... VI

1 INTRODUCTION ................................................................................................................................... 1

2 BACKGROUND ...................................................................................................................................... 3

3 RELATED WORK ................................................................................................................................. 5

4 RESEARCH METHODOLOGY ............................................................................................................. 7
   4.1 RESEARCH METHOD SELECTION ................................................................................................. 7
   4.2 LITERATURE REVIEW ..................................................................................................................... 7
   4.3 SYSTEMATIC LITERATURE REVIEW ............................................................................................. 9
   4.4 CASE STUDY .................................................................................................................................. 9

5 SYSTEMATIC REVIEW ....................................................................................................................... 10
   5.1 GOAL AND RESEARCH QUESTION ............................................................................................... 10
       5.1.1 Aims and Objective .................................................................................................................. 10
       5.1.2 Research Question .................................................................................................................. 11
   5.2 DEVELOPING REVIEW PROTOCOL ............................................................................................... 11
       5.2.1 E-Database and Search Field ................................................................................................ 11
       5.2.2 Search Strategy ....................................................................................................................... 12
       5.2.3 Search String Formulation ..................................................................................................... 13
       5.2.4 Study Selection Criteria ......................................................................................................... 16
       5.2.5 Study Selection Procedure ..................................................................................................... 17
       5.2.6 Study Quality Assessment ...................................................................................................... 17
       5.2.7 Study Extraction Strategy ........................................................................................................ 18
       5.2.8 Review Protocol Evaluation .................................................................................................... 19
   5.3 CONDUCTING SLR ....................................................................................................................... 19
       5.3.1 Identification of Research ......................................................................................................... 19
       5.3.2 Selection of Studies .................................................................................................................. 21
       5.3.3 Quality Assessment .................................................................................................................. 23
       5.3.4 Data Synthesis ........................................................................................................................ 23

6 SYSTEMATIC REVIEW RESULT ....................................................................................................... 29
   6.1 RESEARCH QUESTION 1 ................................................................................................................. 29
       6.1.1 Result ....................................................................................................................................... 29
       6.1.2 Software Testability Factors .................................................................................................... 29
   6.2 RESEARCH QUESTION 2 ................................................................................................................. 32
   6.3 RESEARCH QUESTION 3 ................................................................................................................. 32

7 CASE STUDY ....................................................................................................................................... 41
   7.1 CASE STUDY A: CASE STUDY SETUP .......................................................................................... 41
       7.1.1 Motivation and Alternatives ..................................................................................................... 41
       7.1.2 Methodology ........................................................................................................................... 41
   7.2 PROGRAM DESCRIPTION .............................................................................................................. 42
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.3</td>
<td>EXECUTION</td>
<td>42</td>
</tr>
<tr>
<td>7.4</td>
<td>RESULT</td>
<td>43</td>
</tr>
<tr>
<td>7.5</td>
<td>LIMITATION</td>
<td>46</td>
</tr>
<tr>
<td>7.6</td>
<td>CASE STUDY B: DEPENDENCY EXAMPLE</td>
<td>47</td>
</tr>
<tr>
<td>8</td>
<td>DISCUSSION</td>
<td>48</td>
</tr>
<tr>
<td>9</td>
<td>VALIDITY AND THREATS</td>
<td>52</td>
</tr>
<tr>
<td>9.1</td>
<td>INTERNAL VALIDITY</td>
<td>52</td>
</tr>
<tr>
<td>9.2</td>
<td>EXTERNAL VALIDITY</td>
<td>52</td>
</tr>
<tr>
<td>9.3</td>
<td>CONSTRUCT VALIDITY</td>
<td>52</td>
</tr>
<tr>
<td>9.4</td>
<td>CONCLUSION VALIDITY</td>
<td>53</td>
</tr>
<tr>
<td>10</td>
<td>CONCLUSION</td>
<td>54</td>
</tr>
<tr>
<td>10.1</td>
<td>CONTRIBUTION</td>
<td>54</td>
</tr>
<tr>
<td>10.2</td>
<td>LIMITATION</td>
<td>54</td>
</tr>
<tr>
<td>11</td>
<td>RECOMMENDATION AND FUTURE WORK</td>
<td>55</td>
</tr>
<tr>
<td>REFERENCES</td>
<td></td>
<td>56</td>
</tr>
<tr>
<td>APPENDIX-I</td>
<td></td>
<td>62</td>
</tr>
<tr>
<td>DOCUMENTS USED FOR PILOT, INCLUSION/ EXCLUSION CRITERIA AND KAPPA ANALYSIS</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>APPENDIX-II</td>
<td></td>
<td>66</td>
</tr>
<tr>
<td>QUALITY ASSESSMENT</td>
<td></td>
<td>66</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 4:1: Articles selected for literature review ................................................................. 7
Table 5:1: Research questions and associated goal ............................................................... 11
Table 5:2: Identifying key words and synonyms ................................................................. 14
Table 5:3: Quality Assessment Template ............................................................................ 18
Table 5:4: Data extraction template 1 ..................................................................................... 18
Table 5:5: Data extraction template 2 (for framework analysis) .......................................... 18
Table 5:6: Search result for e-databases .............................................................................. 20
Table 5:7: Number of records exported to Endnote ............................................................. 21
Table 5:8: Number of records for primary selection ............................................................. 21
Table 5:9: Result of Coverage test ....................................................................................... 21
Table 5:10: Kappa coefficient analysis ................................................................................. 21
Table 5:11: List of articles selected for primary study ......................................................... 24
Table 5:12: Author and number of publications ................................................................. 28
Table 6:1: Extracted data based on six different categories .................................................. 34
Table 7:1: Experiment result for participant A ..................................................................... 43
Table 7:2: Experiment result for participant B ...................................................................... 45
Table 7:3: Experiment result for participant C ..................................................................... 46
Table 7:4: Experiment result for participant D ..................................................................... 46
LIST OF FIGURES

Figure 4.1: Research Methodology ................................................................. 7
Figure 5.1: Schematic view of systematic review ........................................ 10
Figure 5.2: Search Strategy ........................................................................ 13
Figure 5.3: Venn diagram representations of search categories ...... 15
Figure 5.4: Study inclusion/exclusion process ...................................... 17
Figure 5.5: Studies by publication year .................................................. 24
Figure 5.6: Distribution of article .............................................................. 28
Figure 6.1: Factors that affect runtime testability .................................. 31
Figure 6.2: Testability Fishbone ................................................................. 32
Figure 6.3: Testability definitions and concepts ........................................ 33
Figure 6.4: Proposed testability assessment framework ..................... 39
Figure 7.1: System description of PD program ....................................... 42
Figure 7.2: Testability tree for difficult to specify inputs .................. 43
Figure 7.3: Testability tree for difficult to interpret output .................. 44
Figure 7.4: Testability tree for difficult to trace internal variables ........ 45
Figure 8.1: Software Testability Factors .................................................. 48
Figure 8.2: List of Factors affecting software testability ................. 48
Figure 8.3: List of Testability metrics ...................................................... 49
Figure 8.4: Domains of a test suite .......................................................... 50
1 INTRODUCTION

In current software development industry, despite knowing the advantage as well as necessity of delivering quality product in market, expected level of quality are becoming more challenging and crucial. With the increasing complexity of software applications, software industry are lack to deliver the quality product and even some time some quality attributes are neglected [61]. In this current highly competitive software market, companies are often trying to meet the release dead line that generally reduces the testing time. Hence the software product may not be properly checked for the potential defects. So we cannot take lightly the quality assurance part of each software product. Fault prevention and fault detection have to be considered in every possible step. Nowadays, testing activities are also under priority so that it become easy and effective to find and treat bugs.

Software testing is one of the crucial activities in the software development life cycle. It is an verification and validation process which aims to expose software faults by executing software product, program or application and do check whether the produced output works as expected or not in order to check the correctness of the stuff. Software testing is an extremely essential means of detecting the software fault [62]. It is also well known reality that more than 50% of the total software development costs is related to the software testing activities [63]. Hence it is one of the most expensive phases of software development life cycle in terms of money as well as time. So it is always the challenging research area in reducing the cost of testing. Many researchers have focused their study for the solutions to minimize the testing cost. If the testability of software can be improved, then it is possible to reduce the software development cost along with achieving the higher easiness in writing test cases, test automation, fault detection.

The testing process is becoming more complex day by day as the software product, process is becoming larger and complex now. Due to the increasing complexity of the product, more new challenges are facing in the testing process and it is also becoming complex. Every testing process has some testing plan which needs to be completed thoroughly in order to detect all defects that the system possesses. But in case of not completing the testing plan, there will hide critical errors that gives unacceptable errors which will give problem for the system. With number of researches devoting towards the testing phase, some of the researches have been really focusing towards the software testability as well. Main intention will be making the testing process easy and detecting the defects in effective, confident way. With the increasing value of testability in software, it is easier to occur the incorrect output in case of presence of defect in the software [12]. Hence measuring the software testable research about their testability factors and improvement the testing process with considering testability can be the important research in current time though there are lots of studies regarding this area in both academic and industrial way. The testability approach increases the probability of revealing the faults ultimately making software fault detection process easier.

The effectiveness of software testing depends upon the quality of the generated test cases. Effectiveness is a measure of degree of extent set goals have been achieved (“doing the right things”). Here in software development terminology, the test effectiveness is described in terms of the test cases and the defects. The test effectiveness can be measured as the ratio of the number of defects found to number of test cases executed. Good test cases can decrease both the workload and the cost of software testing, and can step up the software development without compromising the quality of the software testing [64]. To find such field of research for improving the effectiveness of testing process, we have chosen the “software testability” as a challenging issue to be investigated in the software development process. The considered issue will be analyzed in order to make the software development process
towards simplified testing process and ultimately will be targeted for the better effectiveness of the testing process as a whole [12].

Software testability is defined by IEEE as “the degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met” [65]. ISO has defined software testability as a functionality and it defines functionality as “the set of attributes of software that bear on the effort needed to validate the software produced” [10][66]. From the concepts about the software testability given by different authors, it can be in general defined as the degree to which a software artifact facilitates process of testing. Software testability not only indicates the test process effectiveness but gives new perspective on code development [51].

Software that is possibly revealing the faults within itself during testing is said to have high testability whereas software with low testability do not likely reveal faults during testing [51].

After the first research on the software testability done in 1975, there are other follow- researchers who have contributed in their own way for software testability [67]. Also since 1990s, quantitative research has been carried on the testability by the software engineering community. Mostly the researches have been done on the software testability measure as well as design for testability and others areas.

The major concern of this research will be thorough study of software testability with its factors and metrics implementation of testability keeping in mind to make the test suite generation simpler so that we will have a better effectiveness of the testing process. The information we gained through the systematic review will be used to construct a conceptual framework that will give the guidelines for the test engineers to make the testing process effective. With this guideline, the test engineer will make test plans to create the test cases that fits into the working environment.
2 BACKGROUND

This chapter summarizes the definitions of key terms and concepts that are relevant to our thesis project. The key terms are software testability concept, software testability factors and measurement, framework analysis, test suite generation concept.

Software testing is the process of evaluating features of software product. Software under test (SUT) is analyzed to identify the differences between actual and expected behaviors of the product under specified conditions and based on which the evaluation is performed. IEEE defines it as "An activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component"[65].

Similarly, some other terminologies under the software testing area defined by IEEE [65] are:

- **Mistake**: "A human action that produces an incorrect result."
- **Error**: "The difference between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition."
- **Fault**: "An incorrect step, process, or data definition in a computer program."
- **Failure**: "The inability of a system or component to perform its required functions within specified performance requirements."
- **Test case**: "A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement."
- **Acceptance criteria**: "The criteria that a system or component must satisfy in order to be accepted by a use, customer, or other authorized entity".

In this report, these definitions will support to understand the terminologies used throughout [65].

Testing is well known and applied verification technique of executing a program in order to check the presence of faults. Software testing can never be used to apply to claim show the absence of the faults. From the starting of the software development process, quality assurance is applied on different level of development process. Testing is taken as an investigating process to validate and verify the product meets the specified requirements.

Testing is broadly used in software industry for quality assurance.

The levels of the testing can be categorized as follows. Levels of testing [68]

- **Unit testing**: A unit is smallest testable piece of application software. Unit testing can be defined as the testing process which shows the each unit does satisfy the functional specification and/or that its implemented structure does not match the intended design structure [68]. Unit testing allows taking the smallest testable piece of software in the application, isolating that particular code from the rest of the code and checking whether it functions as intended.

- **Component testing**: A component testing is concerned with testing the one or more units of a component and it is tested with respect to the same criteria as units.

- **Integration testing**: Although each component is tested, there may introduce inconsistencies among the components. The integration testing tests the interface between two already tested units which are combined into a component.

- **System testing**: System testing is intended to test the system’s compliance with its specified requirements on the entire system.

Software testing is an extremely essential means of detecting the software fault [20]. It is also well known reality that more than fifty percent of the total software development costs is related to the software testing activities [9]. So it is always the challenging research area to reduce the testing cost. If the testability of software can be improved, then it is possible to reduce the software development cost along with achieving the higher easiness in writing test cases, test automation, fault detection.
The effectiveness of software testing depends upon the quality of the generated test cases. Effectiveness is a measure of degree of extent set goals have been achieved ("doing the right things"). Here in software development terminology, the test effectiveness is described in terms of the test cases and the defects. The test effectiveness can be measured as the ratio of the number of defects found to number of test cases executed. Fine test cases can decrease both the workload and the cost of software testing, and can step up the software development without compromising the quality of the software testing [16].

To find such field of research for improving the effectiveness of testing process, we have chosen the “software testability” as a challenging issue to be investigated in the software development process. The considered issue will be analyzed in order to make the software development process towards simplified testing process and ultimately will be targeted for the better effectiveness of the testing process as a whole [11]. So, the major concern of this research will be thorough study of software testability with its factors and metrics implementation of testability keeping in mind to make the test suite generation simpler so that we will have a better effectiveness of the testing process.

The concept of testability of a software component was initiated by Freedman [72] considering controllability and observability. The fact about the testability is that poor testability drives to ineffective testing and ultimately severe penalties. If we take the motivation for this work from [19], Beizer writes there is lack of clear testability metrics in practice, but there are only the complexity metrics which gives more bugs and more testing to reveal them. This fact motivates to investigate and explore the complete, robust testability metrics which are practically implementable in industry to make the bug finding process simpler. There is possibility of reducing the development cost along with achieving the higher effectiveness in writing test cases and bug identification [11][12]. But, due to insufficient knowledge on assessment of testability and test case creation considering testability, such achievement is limited in current practices. So, we are motivated to investigate this particular area of testability and test case creation to mitigate this gap and limitation as far as we can.

Therefore, the purpose of this thesis is to understand and use the testability in detail which will lead to simpler test suite generation. For this, the knowledge gained through the research will be used to construct a conceptual framework for test suite generation integrating the testability assessment to help the test engineers in testing process. This framework is aimed at providing scope, purpose and how to perform a testability assessment in test analysis of requirement specification, system design and coding through guidelines. The guideline will also make the test assertion (test description) processes easier, clear and complete. Thus, the reader can extend and adopt testability concept into his/her test plan in order to create the test cases (and test suite) that fits into the working environment.
3 RELATED WORK

A number of testability theories have been published till date and the testability concept has been grown with different research states. Here we will discuss some of the important theories given by some researches in their paper and we will relate those all research with our thesis in order to motivate our work. Numerous studies below give some idea about the related work on this area.

Binder defines software testability as the relative ease and expense of revealing software faults i.e., the software sensitivity to faults [7]. Binder offers an analysis of the factors which are contributing to the software testability. He claims that testability of an object-oriented system, in broad sense, is a result of six primary factors:

- Characteristics of the representation
- Characteristics of the implementation
- Built-in test capabilities
- The test suite
- The test support environment
- The software development process

Binder also listed some of the testability metrics from encapsulation metric, inheritance metric and polymorphism metric. Encapsulation metric covers LCOM (Lack of Cohesion in Methods), PAD (Public Access to Data members) while Some of the inheritance metric are NOC (Number of Children), DIT (Depth of Inheritance Tree) and similarly Polymorphism metric include OVR (Percentage of non-overloaded calls), DYN (Percent of dynamic calls) etc. He also said that test scaffolding (test drivers and stubs) improve controllability and observability which will ultimately increase the software testability.

Voas and Miller in [52] define software testability as the probability that a piece of software will fail on its next execution during testing, provided it contains a fault. They have suggesting dynamic technique called software sensitive analysis for evaluating the software testability. It measures the improvement in software testability by quantifying behavioral information about the likelihood that faults are hiding. Sensitivity analysis repeatedly executes the original software and mutant versions of its source code and estimates the likelihood that those mutants are found out. With the prediction from sensitivity analysis, a tester can easily get the information about even the smallest faults and can concentrate the effort on that particular location. They also suggest the relation between information loss and the testability. Information loss which can be considered as the phenomenon of occurring when internal information computed by the program during execution is not communicated in the program’s output. Software testability decreases with the information loss increases as there is greater potential of cancelling the data-state errors. There is proposing of dividing the information loss into two broad classes: implicit information loss and explicit information loss. Implicit information loss occurs when two or more different incoming parameters are presented to a user-defined function or a built-in operator and produce the same result. They have suggested the “domain/range ratio” metric for a degree of implicit information loss. Explicit information loss observed with static code inspection.

Bache and Mullerburg [69] define testability as a quality attribute to the testing effort needed. The number of test cases (or test runs) needed for satisfying a test criterion is an important factor of the testing effort. They define testability for the control-flow-based test strategies as the number of paths (or test cases) required by the strategy to provide total test coverage, assuming that such coverage is possible. More precisely, they took testability as the minimum number of test cases to provide total test coverage, assuming that such coverage is possible. The measurement of testability has been defined w.r.t. control-flow-oriented test strategies which can be determined by measuring the flow graph. Fenton-Whittyflowgraph theory has been basic consideration.
for the flow graph. Still the authors discussed some problems with testing strategies with insufficiency test strategies with branch testing and simple path testing. The approach is limited to the control-flow based testing strategies.

Baudry [70] mainly concerned with the testability relation to the testing effort focusing on object oriented static designs bases on UML (Unified Modelling Language) class diagrams. The study mainly focused on the testability weaknesses of UML class diagrams. Paper approximates the number of class interactions from UML class diagram, which can be applied to estimate the testing effort and the design testability and it also suggest for the improvement for the class interactions regarding reduction in number and complexity. The author gave a model, the class dependency graph, using UML class diagram features in order to evaluate the complexity of class interactions which is used for the testability measurement. Still the model is not clear about the cause–effect relation between metric and testability.

The author [71] presents the empirical study of the instrumentation of contracts which are used to increase the testability and contracts are the useful technique for specifying precondition and post condition of operations and class invariants. There is some detail study of impact of simplifying on the diagnosability and observability. It also has investigation for the undetected failures from their low diagnosability. It clarifies that contract assertions can reduce the number of methods and source code statements we have to consider in order to position faults in case of failures detection in the system. The systematic investigation of impact of contract precision on observability and diagnosability at three different precision levels are discussed in the paper.

The research by Bruntink and van Deursen [11] is mainly concerned with identifying and evaluating the factors of testability in object oriented software and metrics related to the factors, which are been supported by the case studies. The authors used the source code analysis for characterizing the software testability. In the papers, authors identify possible relevant metrics to predict the class testability and analyzed with hypothesis. The hypothesis was been investigated and tested with the statistical analysis.

Here in this report, author [72] defined the new concept of domain testability of software applying the concepts of observability and controllability. According to report, domain testable software must be observable and controllable. They did an experiment in order to address the advantage of domain testability. They made the conclusion in the report that domain testable program does not exhibit any input-output inconsistencies and they also defined the metrics for accessing the effort level to make the domain testable.
4 RESEARCH METHODOLOGY

This chapter discusses the different research methods that have been chosen to answer the research questions specified in chapter 1, section 1.2. The flow of this section is organized in four sub sections. Section 4.1 describes the Research Method section, section 4.2 describes about the literature review, section 4.3 describes about the Systematic literature review and the last section 4.4 describes about the case study.

Further, how these methods will help to answer the research questions of our thesis. The motivation behind the selection of the research methods are also discussed in the sub sections of this chapter.

4.1 Research Method Selection

As shown in figure below, flow of the methods to get the answer for the research questions are given. The literature review will be used to collect key documents for our research topics. The literature review is conducted for the broad view on the research topics. The motivation of selection of research methodology is provided in section 4.3.

4.2 Literature Review

IEEE and Scopus, two databases are taken to retrieve the key documents. There are two search strings (Software Testability Measurement, Software Testability Factors) which are separately executed in each two databases and returned documents are recorded. As from the table above, the returned documents are mentioned. After that the returned documents from two different search strings are merged and duplicates are removed and thus altogether there are 131 unique documents from Scopus. Similarly there are 89 unique documents from IEEE.

Table 4.1: Articles selected for literature review

<table>
<thead>
<tr>
<th>Database</th>
<th>Search Strings</th>
<th>Returned Documents</th>
<th>Total Documents</th>
<th>After removing duplicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE</td>
<td>Software Testability Measurement</td>
<td>61</td>
<td>95</td>
<td>89</td>
</tr>
</tbody>
</table>
During the proposal writing, Authors have referenced many of the documents related to our research and we found some of these documents greatly relevant to our research area. Hence, we took some of the documents from the proposal references which are listed below.

In order to find out the key documents from the IEEE and Scopus, initially we grouped the total returned documents into clusters with covering all the distribution years and from each clusters, we picked the random numbers which will be our key documents. The key documents here used as a term documents picked in order to validate our search string during systematic review and we have the assumption that the returned documents from the search string which contains the main exact keywords from research questions have the equal potential of being relevant documents. So the randomly picked documents from the population are also the relevant documents. This is achieved with defining population, giving the separate ID for the documents and selecting random number of the samples on the basis of random number generated. It is widely accepted assumption by the Researchers that the random sample represents whole relevant documents. We have altogether 131 unique documents and 7 of these documents are taken as the key documents which are picked randomly using the random number generator methods. The sample size of 7 is taken with the help of statistics. This number was calculated based on the sample size calculator with the confidence level of 95%, Confidence Interval of 37% within given population. This is totally the random methods of selecting key documents. Similar strategy has been done with the IEEE database to retrieve 5 key documents. The retrieved key documents from the two databases are below mentioned.

**From Scopus:**
1. AUTOTESTCON '80 (PROCEEDINGS), 1980. AUTOTESTCON (Proceedings)
2. FUNCTIONAL ATE - VERSATILITY AND LOW COST REQUIRES IN HOUSE DESIGNS, 1986, Luffman, Fredrick E.

**From IEEE:**
1. Measuring and improving design patterns testability
2. Testability Models for Structured Programs

**Key Documents from Proposal:**

4.3 Systematic Literature Review

A systematic review is a means of identifying, evaluating and interpreting the available research findings relevant to a particular research area or topic [73]. The study in research area has broadly divided into two categories primary and secondary studies. Primary study is an individual studies contributing to the research and secondary study is a systematic review of other research related to the research area, topic or phenomenon of interest [73]. The reasons for choosing systematic literature review as methodology of study are to summarize the existing body of knowledge about the research of interest, to identify the gap in current research and to provide framework/background for further investigation. In this context, we choose the systematic review to summarize the existing concepts of testability factors and measurement in software engineering and use that knowledge to develop a testability assessment framework for test suite generation which helps to test engineers for self assessment. The reasons for selecting this methodology are:

1. Systematic literature review's well defined methodology helps to reduce the bias for selecting primary studies.
2. Its systematic process enables consistency in study selection and quality assessment of primary studies.
3. Its outcome serves as input for further framework construction.

The systematic literature review has the following steps [73].

1. Data source selection
2. Search strategy development
3. Search string formation
4. Study selection criteria identification
5. Study quality assessment identification
6. Study extraction strategy identification

Details of each step are presented in the chapter five.

4.4 Case Study

The case study will be performed to study the feasibility of the framework in a real practice. The environment for the case study will be the debugging of a program developed and tested for the faults in a master thesis program at Blekinge Institute of Technology. The selection of the program will be based on that the environment can represent a real environment. The developer of the program and students from the same field will participate in the case study. The cases will be selected on the bases that will closely relate to the problem for which the framework is being proposed. Details of the case study will be present in section 7 of the report.
5 **SYSTEMATIC REVIEW**

This chapter describes the systematic literature review in details. Section 5.1 maps the research questions with specific goals to find out the answers of the research questions. Section 5.2 presents the explanation of review protocol. Section 5.3 presents the data synthesis or analysis planning and section 5.4 shows the details of how systematic review was conducted.

![Schematic View of Systematic Literature Review](image)

**Figure 5.1: Schematic view of systematic review**

### 5.1 Goal and Research Question

The research questions for the systematic review are taken from the research question formulated in proposal. Systematic literature review is focused to answer the each research question based on the facts identified in the literatures included in systematic review. The goal of the each research questions are set so that aims and objectives of the research can be achieved at the end.

### 5.1.1 Aims and Objective

The main aim of the research is to propose a conceptual framework taking testability issues for simpler test suit generation to facilitate the test engineers with better effectiveness of testing.
The scope of the research is confined into testability issues, metrics and its relation to test suite generation process to support the testers. The aim will be accomplished by following objectives.

- Understanding of the term testability in software development process.
- Investigating the metrics to measure testability.
- Understanding co-relation between testability and test suite generation.
- Investigating the evidences to know how test suit generation becomes simpler using testability measurement.
- Analysis of how the evidences support in constructing a framework that works with test suite generation for acceptance testing, system testing and unit testing taking the testability issues.

5.1.2 Research Question

The table below shows the overview of list of research questions that are answered. Each research question has targeted goal and related methodology; tools are also mentioned in this table.

Table 5:1: Research questions and associated goal

<table>
<thead>
<tr>
<th>Research Questions</th>
<th>Goal</th>
<th>Methodology</th>
<th>Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ1: What are the factors of testability identified in current researches?</td>
<td>To identify factors that affect software testability</td>
<td>Systematic Literature Review</td>
<td>MS Word, MS Excel, Endnote</td>
</tr>
<tr>
<td>RQ2: What are the available measurements used to measure factors from RQ1?</td>
<td>To identify how those characteristics are measured and expressed.</td>
<td>Systematic Literature Review</td>
<td>MS Word, MS Excel, Endnote</td>
</tr>
<tr>
<td>RQ3: How do the measurable testability factors from RQ1 and RQ2 contribute to make the test suite generation simpler?</td>
<td>To identify which characteristics and its state present in the different software artifacts affects the test analysis in the software development.</td>
<td>Framework Analysis</td>
<td>MS Word, ATLAS TI 6.0</td>
</tr>
</tbody>
</table>

5.2 Developing Review Protocol

Systematic review protocol provides the formal and structured way of steps that should be followed in the review process. Main aim of the protocol is to define the each and every steps of the systematic review in detail so that the result can be reproducible, explicit and unbiased. This section presents data source, search strategy, search string, study selection, quality assessment, data extraction strategy and criteria of evaluation of these processes when necessary.

5.2.1 E-Database and Search Field

To select the electronic databases, we studied the information available in database list of ITS Learning. We also conducted a literature review to identify the list of databases used by other researchers for systematic reviews. Based on these studies we selected Scopus, EI village2 (Inspec&compendex ), IEEE Explore, SpringerLink and ACM Digital Library. These databases have large number of journal and conference article in the field of software engineering and computer science.
We used the advance search field to conduct the search. The search is limited to find key words in the Title/Abstract/Key words.

In our systematic review, we have chosen the well known and popular database for the software engineering research named IEEE, Engineering village (Compendex, Inspec), ACM, SpringerLink, and Scopus. The selection of this database among many databases can be motivated in this way.

**IEEE**

IEEE Xplore is one of the powerful online resources for full text documents with highly cited journals in the scientific and technical publications managed by the institute of Electrical and Electronics Engineers (IEEE) and its partners. IEEE has over two million articles covering journals, conference proceedings and technical standards. Importantly have to mention is that we can access real time access to updated databases.

IEEE is undoubtedly most-cited publisher in new patents as given by latest studies [80]. Since the articles published in the IEEE are benefited by the peer review process, we can ensure the quality information published in the IEEE database.

**Compendex**

Compendex is one of the most comprehensive bibliographic databases with over 12 million records from 70s to present date. It covers around 190 engineering disciplines and it is fact that over 9, 00,000 records each year and it is the database that is updated weekly [81].

It is trusted resource for thousands of institutions around the globe with depth coverage of engineering research. It includes peer-reviewed journals and it is believed to contain hard to find conference papers that may not available anywhere else.

Also the features provide with this database for searching provide the better option for selecting this database among others.

**Inspec**

Due to different benefits by using inspec, this leading bibliographic database is included in our database list for our systematic review. It covers over 11 million records with competitive intelligence. It also features the data mining concept for analyzing data by a wide variety of parameters.

**ACM**

This is also one of the well trusted resources for citations and full text from ACM journal and newsletter articles and conference proceedings.

**SpringerLink**

SpringerLink is integrated databases that offer full text for journals, books, protocols, e-references and book series. It includes 2436 fully peer-reviewed journals and around 40,153 books online. Beside it also offers free access to search tables of content, abstracts and alerting services [82].

**Scopus**

This database is included due to its quick, easy and comprehensive support for the literature research process and it is easiest way to find relevant results quickly. It contains over 15,000 peer reviewed papers from more than 4,000 publishers. Mainly since it covers all the major peer reviewed titles, it is one of the resources for the systematic review.

5.2.2 **Search Strategy**

The key words are identified from research questions, key words from literature, synonyms and thesaurus. Search string is written and piloted in the databases. Then, duplicates are removed and coverage of search string is check with the key documents. The process is iterated until all key documents are found by the search string. All these search results are documented with keeping log file. We have also consulted the librarian during the searching process techniques as well as for the search terms construction process. The process is shown in the figure.
5.2.3 Search String Formulation

In order to include all the high quality relevant publications related to our research questions, our search strategy for the systematic review can have great play. Keeping mainly in mind the research publications related to software testability, we have followed the systematic steps to make the search string.

1. The keywords are picked up from the each research questions.
2. The keywords are selected from our own knowledge related to this domain and also from the previously done related systematic review in this domain.
3. The synonyms, abbreviations, related terms, variant spelling and the alternative spellings for the major keywords are identified with the help from dictionary.
4. When needed the facility of truncation and wildcards are also been implemented.
5. A pilot search is done in the predefined databases to validate as well in order to discover other keywords, synonyms and alternative spellings as well.
6. The keywords, synonyms, alternative spelling, abbreviations etc are grouped as categories to ease in making the search string.
7. Boolean operator “OR” and “AND” are used to make the refined search string. The Boolean OR is used to join alternative spelling, synonyms whereas Boolean AND to join major search terms.
### Table 5:2: Identifying key words and synonyms

<table>
<thead>
<tr>
<th>Category</th>
<th>Search Terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Factors</td>
<td>Component OR Element OR Ingredient OR Constituent OR Part OR Cause OR Item OR Issues OR Terms OR Means OR Feature OR Checklist</td>
</tr>
<tr>
<td>Testability</td>
<td>Maintainability OR Controllability OR Observability OR isolateability OR understandability OR automatability OR heterogeneity OR test analysis OR test quality OR test designs OR dependencies</td>
</tr>
<tr>
<td>Software</td>
<td>Software OR Application OR Product OR Development OR Project</td>
</tr>
<tr>
<td>Requirement analysis</td>
<td>Requirement OR Requirement Engineering OR Requirement process OR Requirement Phase OR Requirement Study</td>
</tr>
<tr>
<td>Design</td>
<td>Software Design</td>
</tr>
<tr>
<td>Coding</td>
<td>Coding OR Code OR Source Code OR Program Code OR Software Code OR Product Code</td>
</tr>
<tr>
<td>Measurement</td>
<td>Metrics OR Measurement OR Measuring OR Assessment OR Estimate OR Evaluate OR Magnitude OR Valuation OR Value OR Coverage OR Estimation OR Measure OR Evaluation OR Calculation OR Quantification OR Sum OR Amount OR Control OR Range OR Size OR Measurable OR Examination OR Inspection OR Investigation OR Rating OR Survey OR Test OR Valuation</td>
</tr>
<tr>
<td>Testing</td>
<td>Analyze OR Examine OR Assess OR Check OR Confirm OR Demonstrate OR</td>
</tr>
</tbody>
</table>

### 5.2.3.1 Main Search Category

The search area is identified by breaking down the research questions 1 and 2. From the research question 1, the following three search areas are identified.

1. Software engineering
2. Software developing process or software development life cycles.
3. Factor(s) of software testability.

From the research question 2, the following research area is identified.

1. Measurement(s) of software testability

From the population, intervention and outcomes following research area is identified:
- Software development processes – requirement engineering, software design and software development as population
- Software Testability as intervention
- Effective test case generation (or more specifically, improvements in defect detection analysis)

The search area required to answer the research questions 1 and 2 can be expressed in one single line as “software testability factors and measures in software development processes”. This string completely covers the research questions 1 and 2. The population and intervention also covered except the outcomes. The outcome is not relevant to search because the outcomes will be reached through the synthesis of
answers from RQ 1 and 2 which is clearly explained in proposal too. Thus the final search areas are:
A. Software engineering
B. Software developing processes (requirement engineering, software design and software development)
C. Testability
D. Factor, measures

**str1** = (Software OR Application OR Product OR Project)
**str2** = ((SDLC) OR (Requirement AND (Engineering OR process OR Phase OR Study OR Analysis)) OR (Design) OR (Code AND (Source OR Program OR Software OR Product)) OR (testing))
**str3** = (Testability OR Testable)
**str4** = ((Factor OR Component OR Element OR Ingredient OR Constituent OR Part OR Cause OR Item OR Issues OR Terms OR Means OR Feature OR Checklist OR Maintainability OR Controllability OR Observability OR isolateability OR understandability OR automatability OR heterogeneity OR "test analysis" OR "test quality" OR "test design" OR dependencies OR Metrics OR Measurement OR Measuring OR Assessment OR Estimate OR Evaluate OR Magnitude OR Valuation OR Value OR Coverage OR Estimation OR Measure OR Evaluation OR Calculation OR Quantification OR Sum OR Amount OR Control OR Range OR Size OR Measurable OR Examination OR Inspection OR Investigation OR Rating OR Survey OR Test OR Valuation)

We conducted the different combination of these search string and selected the final search string as:

((software OR application) AND testability AND (requirement OR specification OR analysis OR analyze OR design OR architecture OR code OR development OR source OR product OR program OR (factor*)) OR component OR element OR constituent OR issues OR feature OR checklist OR maintainability OR controllability OR observability OR isolateability OR understandability OR automatability OR heterogeneity OR test OR testing OR quality OR case OR dependencies OR metrics OR (measure*) OR assessment OR (evaluat*) OR magnitude OR (value*) OR coverage OR (estimate*) OR calculation OR quantification OR examination OR inspection OR (investigat*) OR rating OR survey OR parameter))

Figure 5.3: Venn diagram representations of search categories
5.2.4 Study Selection Criteria

**Inclusion/Exclusion**

During searching the databases, large numbers of records were available. Hence to select highly relevant records from that number, we need a certain criteria to select them. Applying the exclusion criteria, inappropriate records will be removed while with inclusion criteria, we selected the records that we need for our study.

As shown in figure below, we first start with the exclusion, inclusion criteria for the whole database retrieved records. We will then complete the pilot search in single database. In the pilot search iteration, we will apply the exclusion; inclusion in records from one database and with the kappa value analysis, the understanding between us is compared. Accordingly, we will refine the inclusion, exclusion criteria and as a whole we contributed to refine the protocol.

**Inclusion Criteria:**
1. Papers with English Language are must to be included in our research.
2. Papers from the software engineering and development are only included.
3. The papers can be publications, technical reports, conferences, journals, workshop or grey literature that addresses the research questions.
4. Papers in which the testability was considered in any phases of software development life cycle.
5. Papers with that include the software testability factors, measurement are in our research papers.
6. Papers reporting the relationship between testability and test suite generation.
7. Papers with relationship between testability and test cases.
8. Any case study, observation, experiments, report data that mark any of our research questions.
9. The papers should be available in full text.
10. Any papers claiming the challenges, issues, problems regarding the software testability are included.

**Exclusion Criteria**
1. Do not relate to software engineering or development field.
2. Do not relate to software testability.
3. Do not have the full text in any specified databases.
4. Testability related studies that are solely on the hardware testability.
5. Publications / Reports which are only available with abstract or a PowerPoint slideshow.
6. Papers which texts are available in language except English.
7. The papers are excluded which focus only with software tool for quality training or learning to give testability.
5.2.5 Study Selection Procedure

The study selection procedure comes when we had records come after searching database with final search string and then applied the inclusion/exclusion criteria. Then we did scan the title and abstract of the paper first. During scanning process, when we found that there are discussions of the research questions in some way or other, then those documents were included in the selection.

During the selection process, sometimes we also gone through the conclusion of the documents in order to decide whether to include the paper or not, if we cannot decide that through title and abstract. While during that process, we excluded those papers if they fall within the criteria of exclusion area. We have already mentioned the number of points in the exclusion criteria and we will follow those points while selection process as well. We did consider the mutual understanding between the reviewers in order to include the papers in the selection process.

5.2.6 Study Quality Assessment

The quality assessment questionnaire is presented below. The quality score is assisted in scale of 1 for “yes”, 0 for “no”, and 0.5 for “partial”.

Figure 5.4: Study inclusion/exclusion process
### Table 5.3: Quality Assessment Template

<table>
<thead>
<tr>
<th>SN</th>
<th>Quality Questionnaire</th>
<th>Assessment (Y/N/P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Is the document credible?</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Are the aims and objectives clearly mentioned?</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Are results of the document related to our research objectives or research questions?</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Is there any software testability factors clearly mentioned?</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Is there any software testability measurements clearly mentioned?</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Is data collection method adequately explained?</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Is the research design suitable in terms of population, intervention and outcomes?</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>How well the collected data is analyzed?</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>How well the diverse perspectives and contexts to software testability and testing have been explored?</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Are presented data complete with well-built argumentation and referencing?</td>
<td></td>
</tr>
</tbody>
</table>

### 5.2.7 Study Extraction Strategy

#### Table 5.4: Data extraction template 1

<table>
<thead>
<tr>
<th>General Information</th>
<th>Data Extracted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Article ID</td>
<td></td>
</tr>
<tr>
<td>Article Title</td>
<td></td>
</tr>
<tr>
<td>Author</td>
<td></td>
</tr>
<tr>
<td>Source</td>
<td></td>
</tr>
<tr>
<td>Date of Publication</td>
<td></td>
</tr>
</tbody>
</table>

Specific information

| Research Methods    |                 |
| Population          |                 |
| Interventions       |                 |
| Outcomes            |                 |
| Problems considered |                 |
| Platform used (if any) |             |
| Development phase considered | |
| Supporting evidences |               |
| Results             |                 |
| Other data          |                 |

### Table 5.5: Data extraction template 2 (for framework analysis)

<table>
<thead>
<tr>
<th>Testability factors</th>
<th>Accountable for Method</th>
<th>Metric SDLC</th>
<th>progranm</th>
<th>Article reference</th>
</tr>
</thead>
</table>
5.2.8 Review Protocol Evaluation

During the systematic review design, systematic review has been refined. The updates in review protocol were made during the search string development, inclusion/exclusion criteria development and data extraction. The evaluation of the protocol was done by carrying out the pilot search for search string. The pilot search result was evaluated by key documents to determine the needs of updated in search string. The inclusion exclusion criteria were updated until the agreement level between the reviewers was not achieved to the acceptable range. The agreement level was calculated by Kappa analysis.

5.3 Conducting SLR

This section covers the details of the processes conducted during the course of systematic review process. The overall review process is conducted in five distinctive and successive processes: identification of research, selection of studies, quality assessment, data extraction and data synthesis [73]. The intermediate results of these processes are also presented in the following subsections successively.

5.3.1 Identification of Research

To identify the research papers, five electronic databases are selected. Selection of these databases is based on:

1. Peer reviewed articles (conference and journal) in the field of software engineering and computer science.
2. Accessibility to both Meta data and full text.
3. Up to date and latest articles are available.

The final search string specified in the protocol was executed in five databases separately. During the execution, some databases did not support the exact string. For example, IEEE database has restriction to use more than five truncations in a search string; SpringerLink has the restriction on the size (no of character in search strings) and ACM does not support for the AND, OR etc operators in the string rather provide the separate fields for those operator (i.e. we can list the keywords for OR in one field and list of keywords for AND in another field). Therefore, the main search string was executed in EI Village and Scopus and some modifications were made for other databases. The search results were exported to local endnote library with complete meta data available.

**Search string for EI Village and Scopus**

```
(software OR application) AND testability AND (requirement OR specification OR analysis OR analyze OR design OR architecture OR code OR development OR source OR product OR program OR (factor*) OR component OR element OR constituent OR issues OR feature OR checklist OR maintainability OR controllability OR observability OR isolateability OR understandability OR automatability OR heterogeneity OR test OR testing OR quality OR case OR dependencies OR metrics OR (measure*) OR assessment OR (evaluat*) OR magnitude OR (value*) OR coverage OR (estimate*) OR calculation OR quantification OR examination OR inspection OR (investigat*) OR rating OR survey OR parameter))
```
Search string for ACM

AND field:
Software testability

OR field:
Requirement specification analysis analyze design architecture code development source product program factor* component element constituent issues feature checklist maintainability controllability observability isolateability understandability automatability heterogeneity test testing quality case dependencies metrics measure* assessment evaluat* magnitude value* coverage estimate* calculation quantification examination inspection investigat* rating survey parameter

Search string for Springer link

((software or application) and testability and (requirement or specification or analysis or analyze or design or architecture or code))

Search string for IEEE

((software OR application) AND testability AND (requirement OR specification OR analysis OR analyze OR design OR architecture OR code OR development OR source OR product OR program OR (factor*)) OR component OR element OR constituent OR issues OR feature OR checklist OR maintainability OR controllability OR observability OR isolateability OR understandability OR automatability OR heterogeneity OR test OR testing OR quality OR case OR dependencies OR metrics OR (measure*) OR assessment OR (evaluat*) OR magnitude OR (value) OR coverage OR (estimate*) OR calculation OR quantification OR examination OR inspection OR (investigat*) OR rating OR survey OR parameter)

Table 5-6 shows the total number of hits found on the respective databases. The search results include the articles published to the end of 2010.

<table>
<thead>
<tr>
<th>SN</th>
<th>Database name</th>
<th>Search type</th>
<th>Auto stemming</th>
<th>No. of hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ei Village 2(Inspec + Compendex)</td>
<td>Expert search</td>
<td>ON</td>
<td>2535</td>
</tr>
<tr>
<td>2</td>
<td>Scopus</td>
<td>Advance search</td>
<td>-</td>
<td>6252</td>
</tr>
<tr>
<td>3</td>
<td>IEEE</td>
<td>Advance search</td>
<td>-</td>
<td>2551</td>
</tr>
<tr>
<td>4</td>
<td>ACM</td>
<td>Advance search</td>
<td>-</td>
<td>2196</td>
</tr>
<tr>
<td>5</td>
<td>SpringerLink</td>
<td></td>
<td></td>
<td>2415</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td></td>
<td></td>
<td>15949</td>
</tr>
</tbody>
</table>

Table 5-7 shows the total number of documents (including Meta data available) exported to local endnote library from the pool of returned documents. The exclusion criteria used to exclude the returned documents are also present in the inclusion/exclusion criteria for the study selection. Also, detailed exclusion criteria were not used. The articles from absolutely irrelevant field such as medical, civil engineering, physics, chemistry, social science etc were excluded. The articles published in other languages (i.e. not written in English) were also excluded. The reasons behind the exclusion were to make the documents export process faster and also to meet the restriction criteria enforced by the databases. Although, the exclusions were made completely using the facilities available in the databases, the original search results were saved in the data based creating the user accounts in respective databases if possible.
Next step in identification of research was to find the duplicated copies of the studies. The duplicated copies were identified using the “find duplicates” function in the endnote local library. The total duplicated articles were found to 1712 out of 5719 articles. From the duplicated pool of articles 253 unique articles were found. Thus, the total articles found for the further selection were 4260. The data are presented in the following table 5-8.

Table 5:7: Number of records exported to Endnote

<table>
<thead>
<tr>
<th>SN</th>
<th>Database name</th>
<th>No. of articles exported to endnote library</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EI Village (Inspec+ Compendex)</td>
<td>2242</td>
</tr>
<tr>
<td>2</td>
<td>IEEE</td>
<td>1932</td>
</tr>
<tr>
<td>3</td>
<td>Scopus</td>
<td>1113</td>
</tr>
<tr>
<td>4</td>
<td>SpringerLink</td>
<td>251</td>
</tr>
<tr>
<td>5</td>
<td>ACM</td>
<td>170</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>5708</td>
</tr>
</tbody>
</table>

At the end of the process, the coverage of the search is carried out. From the pool of 4260 documents, all the key documents were identified. Based on the result, the search process was stopped. The result is presented in the table 5-9.

Table 5:8: Number of records for primary selection

<table>
<thead>
<tr>
<th>Articles exported to endnote library</th>
<th>Duplicate articles</th>
<th>Unique articles in duplicated pool</th>
<th>Total unique articles</th>
</tr>
</thead>
<tbody>
<tr>
<td>5719</td>
<td>1712</td>
<td>253</td>
<td>4260</td>
</tr>
</tbody>
</table>

Table 5:9: Result of Coverage test

<table>
<thead>
<tr>
<th>Key articles</th>
<th>15 (in number)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Articles from search</td>
<td>4260 (in number)</td>
</tr>
<tr>
<td>Identified key articles</td>
<td>15 (in number)</td>
</tr>
<tr>
<td>Coverage of search</td>
<td>100% (in percentage)</td>
</tr>
</tbody>
</table>

5.3.2 Selection of Studies

The next level process was selection of studies in which the inclusion and exclusion criteria were used to select the primary studies. To define the inclusion and exclusion criteria, the guidelines from kitchenham [73] were referenced. A pilot process was conducted to refine the detail inclusion and exclusion criteria. For the pilot selection process, 75 articles from Ei Village 2 database were included randomly. Both authors conducted the selection of primary studies and noted the included and excluded number of articles. Then, the Kappa analysis was performed to determine the agreement level for the inclusion and exclusion criteria. The process was repeated until required level of agreement was reached.

The following table presents the result of Kappa coefficient analysis for the pilot process.

Table 5:10: Kappa coefficient analysis

<table>
<thead>
<tr>
<th>Kappa analysis(Initial)</th>
<th>Reviewer 1</th>
</tr>
</thead>
</table>
After defining the detail inclusion and exclusion criteria, the selection process was conducted. The Endnote database was used as the local library where total 4260 articles with meta-data were stored. The Smart Group facility provided in the program was used to divide the articles in different groups. Smart groups were created using the keywords from the completely irrelevant field for example: civil, hydraulics, gasoline, medical. To cross check and insure the relevant documents were not excluded, key words from relevant field (search string) were used within each smart group.

For the remaining pool of documents, detail inclusion and exclusion criteria were applied manually. The Title/Abstract was manually checked and decision was taken according to criteria defined. To ensure that the relevant article was not excluded, the authors reviewed the excluded articles as a cross validation. Finally, the documents for which full text was not available were excluded. The following table shows the data obtained in the process.
5.3.3 Quality Assessment
The quality assessment was performed according to check list referenced to [73]. The details of quality assessment is shown in appendix.

5.3.4 Data Synthesis
This section mainly shows the characteristics of the primary studies obtained from the systematic review. Figure below shows the distribution of articles according to publication year. Figure below show the trends of the research related to software testability. More research has been conducted in the year 2010 as the figure suggests.
All the primary studies obtained from the systematic search are tabulated below with the title, associated authors, publication year and the reference type of the papers. We have obtained the systematic review results for the research questions from these articles. A total of 36 articles were selected for primary studies. Using inclusion and exclusion criteria 66 articles were selected. After doing quality assessment and data extraction only 36 articles were selected for the primary study. The remaining articles were not rejected due to lack of validations and incomplete data based on our research.

Table 5:11: List of articles selected for primary study

<table>
<thead>
<tr>
<th>SN</th>
<th>Study</th>
<th>Author(s)</th>
<th>Title</th>
<th>Journal/Conference/ Workshop</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>SN</td>
<td>Study</td>
<td>Author(s)</td>
<td>Title</td>
<td>Journal/Conference/Workshop</td>
<td>Year</td>
</tr>
<tr>
<td>----</td>
<td>-------</td>
<td>-----------</td>
<td>-------</td>
<td>-----------------------------</td>
<td>------</td>
</tr>
<tr>
<td>SN</td>
<td>Study</td>
<td>Author(s)</td>
<td>Title</td>
<td>Journal/Conference/Workshop</td>
<td>Year</td>
</tr>
<tr>
<td>----</td>
<td>-------</td>
<td>-----------</td>
<td>-------</td>
<td>-----------------------------</td>
<td>------</td>
</tr>
<tr>
<td>SN</td>
<td>Study</td>
<td>Author(s)</td>
<td>Title</td>
<td>Journal/Conference/Workshop</td>
<td>Year</td>
</tr>
<tr>
<td>----</td>
<td>-------</td>
<td>-----------</td>
<td>-------</td>
<td>-------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>29</td>
<td>[P29]</td>
<td>A. Sabane</td>
<td>Improving System Testability and Testing with Microarchitectures</td>
<td>Reverse Engineering (WCRE)</td>
<td>2010</td>
</tr>
<tr>
<td>30</td>
<td>[P30]</td>
<td>Y. Singh and A. Saha</td>
<td>Improving the testability of object oriented software through software contracts</td>
<td>SIGSOFT Software Engineering Notes</td>
<td>2010</td>
</tr>
<tr>
<td>31</td>
<td>[P31]</td>
<td>Jianping Fu and Minyan Lu</td>
<td>Request-Oriented method of software testability measurement</td>
<td>International conference on information technology and computer science</td>
<td>2009</td>
</tr>
</tbody>
</table>
This pie chart in figure 5.6 shows the distribution of articles obtained from systematic review according to the types of the papers. From chart below suggests that more articles have been obtained from the conference proceedings 75 percent of the total primary studies and that of journal articles are 17 percent of total primary studies. 8 percent of the total articles were from workshop.

![Pie Chart](image)

Figure 5.6: Distribution of article

The table 5.12 shows the authors and the number of articles they have published in this research field. The authors who have published up to 2 articles are only shown in the table. J.M.Vaos and K.W. Miller has highest number of publications as compared to other authors. J.M. Vaos has 5 research papers published followed by K.W. Miller with 4 publications.

Table 5:12: Author and number of publications

<table>
<thead>
<tr>
<th>Author</th>
<th>No. of Publications</th>
<th>Author</th>
<th>No. of Publications</th>
</tr>
</thead>
<tbody>
<tr>
<td>J.M.Vaos</td>
<td>5</td>
<td>C.Robach</td>
<td>2</td>
</tr>
<tr>
<td>K.W.Miller</td>
<td>4</td>
<td>M. Bruntink</td>
<td>2</td>
</tr>
<tr>
<td>Y.L.Traon</td>
<td>3</td>
<td>A.Van Deursen</td>
<td>2</td>
</tr>
<tr>
<td>A. Bertoloni</td>
<td>2</td>
<td>Jin- Cheng Lin</td>
<td>2</td>
</tr>
<tr>
<td>L. Stringini</td>
<td>2</td>
<td>Szu- Wen Lin</td>
<td>2</td>
</tr>
</tbody>
</table>
6  **SYSTEMATIC REVIEW RESULT**

This chapter presents the systematic review results of our research. This chapter content will be the main contribution of our thesis work and will answer the research questions.

6.1  **Research Question 1**

RQ1: What are the factors of testability identified in current researches?

6.1.1  **Result**

The factors of testability identified from the systematic review of the primary studies are as follows. In order to get the factors that affect the software testability, systematic review is conducted on the document obtained from main search string and then after inclusion, exclusion and some of the important factors have been listed out obtained from the primary studies documents.

6.1.2  **Software Testability Factors**

**RQ1: What are the factors of testability identified in current researches?**

- Software component testability depends mainly of five factors
  - Component understandability
  - Component observability
  - Component traceability
  - Component controllability
  - Component testing support capability

There are many definitions of the software testability and different papers have given different types of software testability factors and metrics on different perspective. Some of the important software testability factors are tabulated in the table. From the lists on the table we have software testability factors from the different perspective sources. Some study explains software testability as Internal attributes while some explains as external attributes. It is really difficult task to get all the clear view of factors that can affect the software testability with their dominating degree on the software system. Even though, our research has collected some of the factors that affect the software testability from the systematic literature review. Among the factors that affect the testability obtained from systematic review, we have divers view on the software testability understanding. Research like PIE, Fault tree analysis, Control Flow Analysis took software testability as external attributes while DRR, class interaction etc took it as internal attributes. Software testability is related with the testing process with respect to testing effectiveness. Testing effectiveness not only depends upon the particular software but also upon corresponding test tools, context etc, they will also come under the factors that affect software testability.

According to the factors obtained from systematic review that affect the software testability, Basically we can categorize the factors as we can see, factors are runtime testability factors, component testability factors, object oriented system testability factors and other different factors depending upon the particular systems. Design pattern and Anti pattern are the factors that can affect the software testability of Object Oriented system. We can apply refactoring techniques and guidelines for making design decisions so that the system testability also improves at primitive stage.

Component Understandability, Component Observability, Software Component traceability, Component controllability, Component test support capability are all the component related testability factors, each having different level of contribution for the testability on the software components.
Below are some of the factors related to the runtime testability. Runtime testability is basically described by two terms: test sensitivity and test isolation. Test Sensitivity: Here are mainly four factors that have influence over Test Sensitivity.

**Component State:** is an important factor of test sensitivity which determines if the component exhibits some kind of external state.

**Component Interactions:** Runtime testability of a component depends on runtime testability of the components from the system within or interacting external factors that could be outside the boundaries of system.

**Resource Limitations:** are the non functional characteristic for the testability. We may have resource constraints as processor or memory usage, timing constraints, or even power consumption constraints and it must be ensured that resources availability should not affect for the components when they need them.

**Availability:** The availability requirements of the system can also be considered as one of the important factors. If the component need high availability requirement, then runtime testability may be not be expected.

**Test Isolation:** Have four main factors that have influence over it.

**State Separation:** These techniques are the counterpart of state sensitivity which aim to separate the in-service state of a component from the testing state.

**Interaction Separation:** can be useful to component interactions that propagate through the system and affect other components, and, in particular, the external environment of the system.

**Resource Monitoring:** It is one of the factors which must be applied in order to prevent test cases from exhausting the system resources.

**Scheduling:** In order to preserve the availability of the components, test can be rescheduled.

Regarding the software testability measurements, we have the factors that affect the software testability obtained from systematic review. We also collected the relevant testability measurements techniques, methods on the research papers.

In general, Software testability measurement means methods in order to study, analyze and measure software testability. Number of researches have been carried out in past and in present also there are number of studies in order to address the software testability measurement. Among number of testability measurements techniques that depend upon different attributes, we are here basically categorized PISCES, DRR, Extension of SATAN Technology, Size related metrics, Inheritance related metric, Cyclomatic Number and Pentagram Model for the testability measurements. All these measurement are basically fall under program based measurement methods, Model based measurement methods and Dependability assessment methods. The program based testability measurement works basically under the idea of software mutation testing. That means to get the idea of the fault in the software in any location of program, single fault is instrumented in that location and modified program will be observed with all required assumption. Then basic techniques of execution, infection and propagation of failure will be applied to get the probability of that program. Whereas, Model based testability will use data flow model for the testability prediction and Dependency Based testability measurement will use black box approach with dependency relationships between program inputs and outputs.
Factors related to object oriented software systems

1. Structure factors.
   a. Internal testability of a method
      i. Execution rate of a method
      ii. Propagation rate of a method
   b. Number of methods in a class
   c. Cohesion among methods

2. Communication factors

3. Inheritance factors
   a. Depth in the inheritance tree
   b. Number of children
   c. Number of disjoint inheritance trees

**Antipattern:** are weaknesses that reduce the testability of software. The testability antipattern could affect testability efforts intractable and make test ineffective [42].

**Design pattern:** widely comes in object oriented systems and are one of the influencing factors that impact on system testability [40].

According to [7], there are six primary testability factors.

Implementation characteristics determine controllability and observability. Built-in test capabilities can improve controllability and observability and decouple test capabilities from application features. An adequate and usable test suite (test cases and plan for their use) is necessary. Test tools are necessary for effective testing. High leverage is available with an integrated tool set. Without an effective organizational approach to testing and its antecedents, technical testability is irrelevant.

**Testing Criterion:** There will be always trade-off between the validity requirements with the required amount of testing.

**Documentation:** Documents, like requirements and specifications should be correct and complete. There should be always clean links between the concepts within various documents.

**Implementation:** The extent to which the implementation allows itself to be tested is a key factor of the testing effort.

**Test Suite:** There are other factors for the test suite and it determines the effort required to test. Test suite needs documentation for the test plan, test results of test runs and for the detailing of the implemented tests.

**Test Tools:** Proper use of test tools can significantly decrease the required effort.

**Process Capability:** Here we refer process as the organizational structure, staff and resources and they have large impact on the testing effort.

Binder [7] talks about six major factors that, according to him, result in testability in the system development process.
6.2 Research Question 2

RQ2: What are the available measurements used to measure factors from RQ1?

Results from RQ1 list software testability factors. In this question, we will investigate the available measurements to measure the factors.

In general, Software testability measurement means methods in order to study, analyze and measure software testability. Number of researches have been carried out in past and in present also there are number of studies in order to address the software testability measurement.

One of the classifications of software testability measurement as shown in [79] consists three main groups.

- Program based measurement methods for software testability [5]
- Model-based measurement methods for software testability [74]
- Dependability assessment methods for software testability [5]

**Program based testability measurement**: According to [74], J.C. Lin et al. proposed a program based method to measure software testability. It considers single faults in a program. The fault can be injected in any location within program. Thus all locations of the program can be taken into consideration when estimating the testability [74]. The basic idea behind this measurement is based on the mutation testing. To find the software testability in any location of source code, a single fault will be inserted in that location and then the mutant program will be compiled and executed with the input. After that, basic techniques of execution, infection and propagation of failure will be applied to measure the probability of failure at that location.

**Model based Testability Measurement**: In this method of testability measurement, data flow model is applied to predict the software testability. In this approach there are three steps. In step first, the program will be normalized to make testability measurement normalized and precise. In the following step, program’s testable elements will be identified according to normalized data flow model. And in the final step, testability of the program is measured according to data flow testing criteria.

**Dependency Based Testability Measurement**: According to A. Bertolino and L. Strigni [5] for the dependency based testability measurement, a black box testing approach will be used where the software testability measurement is performed with relationships between program inputs and outputs.

6.3 Research Question 3

RQ 3: How do the measurable testability factors from RQ1 and RQ2 contribute to make the test suite generation simpler?

From the RQ 1, the factors considered for analysis of software testability are extracted. Similarly, RQ 2 answered the measurements considered for the analysis of software testability is extracted. To identify the important factors for the test suite
generation, another systematic review is conducted. The systematic review examined the included articles in the six different categories as below:

1. **Testability concepts and definition**
   In this category, we have summarized the core concepts used while analyzing the software testability which were given by researcher in different articles. The definition of software testability can be different for different methods of testability analysis. We have found the seven different concepts and definition for the software testability.

   ![Software Testability- definition and concepts](image)

   - Prediction of tendency for failures to be observed during testing
   - Controllability and observability
   - Based on ISO definition
   - Degree of ease to test program
   - Evolutionary Testability
   - Request Oriented Framework
   - Conflict analysis of concurrent programs

   Figure 6.3: Testability definitions and concepts

2. **Testability definitions and concepts**
   There are different methods to measure the testability explained in the literatures. The methods of testability determine the measurements for factors and metrics for the assessment. The variation of methods are due to the reason that one method used for a context or in one abstraction level are extended to another context or in other abstraction level. In broad category the methods are divided to mathematical/statistical methods and qualitative methods that includes the graphical, set theory, data flow analysis etc. Depending upon the methods of assessment, the testability affecting factors can be different.

3. **Validation of method**
   In this category, we focused on whether the studies done by the researcher have reported the validation of method or not. The validation can be the experiments, case study, survey etc.

4. **Context**
   In this category, we have summarized in which context the studies have been done. The contexts identified from the studies are probability of faults detection in the program, information losses, cost and difficulties in test case generation. The study of the context helps to understand that what the objectives of the testability analysis are.

5. **Pro-activeness and Re-activeness**
   In this category, we have summarized when the methods are used in the SDLC phases for the testability analysis. The pro-activeness represents the category for which the analysis is done in the early stages of SDLC. Category is differentiated on the basis that if the analysis is done after the development of the program, it is re-active. If the analysis can be done before the development of the program, it is reported as pro-active.

6. **Abstraction Level**
In this category, we have summarized abstraction level of the analysis. The abstraction level is defined as requirement level, design level, source code level, and module. The table 6.1 shows the data abstracted from the articles based on the six categories.

Table 6:1: Extracted data based on six different categories

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Concept and Definition</th>
<th>Method</th>
<th>Validation of Method</th>
<th>Context</th>
<th>Pro-activeness/Re-activeness</th>
<th>Abstraction Level (Level of Measurement)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[P1]</td>
<td>Not primary concept</td>
<td>Not primary method, summarize available method wrt SDLC.</td>
<td>No formal validation</td>
<td>SDLC</td>
<td>Both</td>
<td>Design, Coding, Unit test, System test, Reliability Assessment</td>
</tr>
<tr>
<td>[P2]</td>
<td>Prediction of tendency for failures to be observed during testing</td>
<td>PIE</td>
<td>Yes</td>
<td>Fault Detection</td>
<td>Reactive</td>
<td>Coding</td>
</tr>
<tr>
<td>[P3]</td>
<td>Controllability and observability</td>
<td>Method is not Mentioned</td>
<td>No</td>
<td>Program Type (Object Oriented)</td>
<td>Both</td>
<td>Class</td>
</tr>
<tr>
<td>[P4]</td>
<td>Prediction of tendency for failures to be observed during testing</td>
<td>Dependability Analysis</td>
<td>Validated through statistical analysis</td>
<td>Fault Detection</td>
<td>Reactive</td>
<td>Design and code level</td>
</tr>
<tr>
<td>[P5]</td>
<td>Prediction of tendency for failures to be observed during testing</td>
<td>Discriminant Analysis</td>
<td>Yes</td>
<td>Fault Detection</td>
<td>Reactive</td>
<td>Module</td>
</tr>
<tr>
<td>[P6]</td>
<td>Controllability and Information</td>
<td>Yes (Case)</td>
<td>Test case</td>
<td>Both</td>
<td>Design and code</td>
<td></td>
</tr>
<tr>
<td>S. No.</td>
<td>Concept and Definition</td>
<td>Method Validation of Method</td>
<td>Context Pro-activeness/Re-activenss</td>
<td>Abstraction Level (Level of Measurement)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>------------------------</td>
<td>----------------------------</td>
<td>------------------------------------</td>
<td>---------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Observability</td>
<td>Transfer Graph Study)</td>
<td>generati on</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[P7]</td>
<td>Prediction of tendency for failures to be observed during testing</td>
<td>Information loss analysis in program</td>
<td>No Fault Detection</td>
<td>Reactive Code Level</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[P8]</td>
<td>Prediction of tendency for failures to be observed during testing</td>
<td>Dependability Validat e through Statistical analysis</td>
<td>Fault Detec tio n</td>
<td>Reactive Code Level</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[P9]</td>
<td>Degree of ease to test program</td>
<td>DFT (Design for Testabili ty)</td>
<td>Test cost reduction</td>
<td>Proacti ve Specification</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[P10]</td>
<td>Prediction of tendency for failures to be observed during testing</td>
<td>Improve ment in Voas method</td>
<td>No validati on</td>
<td>Fault detectio n</td>
<td>Reactive Code level</td>
<td></td>
</tr>
<tr>
<td>[P12]</td>
<td>Controllability and Observability</td>
<td>Assertio n Placeme nt</td>
<td>Yes (Experiment)</td>
<td>Fault Detectio n</td>
<td>Reactive Code</td>
<td></td>
</tr>
<tr>
<td>[P13]</td>
<td>Prediction of tendency for failures to be</td>
<td>Improve ment in Voas method (Block Level)</td>
<td>Fault Detectio n</td>
<td>Reactive Code (Block level)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[P7] [P8] [P9] [P10] [P11] [P12] [P13]
<table>
<thead>
<tr>
<th>S. No.</th>
<th>Concept and Definition</th>
<th>Method</th>
<th>Validation of Method</th>
<th>Context</th>
<th>Pro-activeness/ Reactivenss</th>
<th>Abstraction Level (Level of Measurement)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[P14]</td>
<td>Controllability and Observability</td>
<td>Data Flow</td>
<td>Yes (Case Study)</td>
<td>Informatio n loss</td>
<td>Proactive</td>
<td>Design</td>
</tr>
<tr>
<td>[P16]</td>
<td>Conflict analysis of concurrent OO programs</td>
<td>Conflict graph</td>
<td>No</td>
<td>Localization of fault</td>
<td>Reactive</td>
<td>Design and code</td>
</tr>
<tr>
<td>[P20]</td>
<td>Degree of ease to test program</td>
<td>Fault tree analysis</td>
<td>Yes (on one module)</td>
<td>Fault Localization</td>
<td>Reactive</td>
<td>Code</td>
</tr>
<tr>
<td>[P21]</td>
<td>Controllability and Observability</td>
<td>Data flow</td>
<td>Test (case Study)</td>
<td>Fault Localization</td>
<td>Reactive</td>
<td>Code</td>
</tr>
<tr>
<td>[P22]</td>
<td>Design for testability (best practice)</td>
<td>Design complexity</td>
<td>No</td>
<td>Requirement coverage</td>
<td>Proactive</td>
<td>Requirement Level</td>
</tr>
<tr>
<td>[P23]</td>
<td>Degree of ease to test program</td>
<td>SOCK model</td>
<td>No (Real Life experience)</td>
<td>Complexity of software component</td>
<td>Proactive</td>
<td>Planning and design</td>
</tr>
<tr>
<td>[P24]</td>
<td>Runtime Testability</td>
<td>Component Interaction Graph</td>
<td>Yes (Experiment)</td>
<td>Fault Localization</td>
<td>Reactive</td>
<td>Design and code</td>
</tr>
<tr>
<td>[P25]</td>
<td>Controllability and Observability</td>
<td>Set theory</td>
<td>Yes (Experiment)</td>
<td>Complexity</td>
<td>Both</td>
<td>Design and code</td>
</tr>
<tr>
<td>[P26]</td>
<td>Degree of ease to test program</td>
<td>Design pattern</td>
<td>No (Explain with Example)</td>
<td>Complexity related to</td>
<td>Reactive</td>
<td>Code</td>
</tr>
<tr>
<td>S. No.</td>
<td>Concept and Definition</td>
<td>Method</td>
<td>Validation of Method</td>
<td>Context Pro-activeness/ Re-activeness</td>
<td>Abstraction Level (Level of Measurement)</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>------------------------</td>
<td>--------</td>
<td>----------------------</td>
<td>----------------------------------------</td>
<td>------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>[P27]</td>
<td>Controllability and Observability</td>
<td>Data flow</td>
<td>Yes (Experiment)</td>
<td>Information Loss</td>
<td>Reactive Module</td>
<td></td>
</tr>
<tr>
<td>[P28]</td>
<td>Controllability and Observability</td>
<td>IGF wrt Scicos environment</td>
<td>Yes (Case Study)</td>
<td>Fault Localization</td>
<td>Proactive Design Level</td>
<td></td>
</tr>
<tr>
<td>[P29]</td>
<td>Degree of ease to test program at design phase</td>
<td>Design Pattern</td>
<td>No</td>
<td>Micro-architecture Design pattern and anti patterns</td>
<td>Proactive Design Level</td>
<td></td>
</tr>
<tr>
<td>[P30]</td>
<td>Based on ISO definition</td>
<td>Static code analysis</td>
<td>Yes (Example based on code)</td>
<td>Dependency</td>
<td>Reactive Code</td>
<td></td>
</tr>
<tr>
<td>[P31]</td>
<td>Request Oriented Framework</td>
<td>User Requests</td>
<td>No (Conceptual framework is given)</td>
<td>Affecting factors</td>
<td>Proactive Design and code</td>
<td></td>
</tr>
<tr>
<td>[P33]</td>
<td>Prediction of tendency for failures to be observed during testing</td>
<td>PIE</td>
<td>Yes (Experiment)</td>
<td>Fault Detection</td>
<td>Reactive Code</td>
<td></td>
</tr>
<tr>
<td>[P34]</td>
<td>Prediction of tendency for failures to be</td>
<td>Extended PIE</td>
<td>Yes (Experiment)</td>
<td>Fault Detection</td>
<td>Reactive Code</td>
<td></td>
</tr>
</tbody>
</table>
The study of testability concepts and definition shows that there is no single definition and concept to analyze the software testability of a system or component. There are different perspectives for the analysis. IEEE defines testability as “the degree to which a system or component facilitates the establishment of test criteria and performance of tests to determine whether those criteria have been met”. ISO, on the other hand defines it as “attributes of software that bear on the effort to validate the software product”. These two definition aims to define the testability in different perspective. IEEE definition focuses on coverage criteria and ISO focuses on the effort required. Similarly, the other authors define their own perspective and establish the measurement framework. Table shows the different perspectives and concepts. The summary of the factors and measurements from RQ 1 and 2 also shows that the factors affecting the testability of the system come from the different perspective taken to analyze.

From the perspective of test suite, we have found two concepts in the literature. One is controllability and observability and another is evolutionary testability. Similarly, other methods focus on the fault detection probability, information losses, dependability etc. On the other hand, statistical and mathematical methods use the complex functions to calculate the testability. The calculation is hard to understand for the novice test engineers. Another point is, the domain of analysis may not cover the factors from those domains for which test suite may need to include the information. For example, PIE technique does not include the test oracle. To make the test case generation simpler and easy to understand for novice testers, there is a need of easy framework in which the concepts from those methods can be used and applied the context of test suite generation.

Freedman [72] has analyzed the testability of a program component. According to him, a testable component had the following characteristic of test sets:

- Small (few in number) and easy to generate.
- Non redundant
- Easy to interpret outputs
- Easy to locate faults

So, any kind of difficulties or anomalies to define a scenario for set of inputs, program output and expected output are the immediate conditions that lead to component or program to be untestable. Freedman, in “testability of software component” paper, has described two criteria to decide whether not the component is easily testable or not. The criteria are input inconsistency and output inconsistency. When a tester foresees that one of these criteria is present in the component then the component is said to be “not testable”. Based on these criteria, Freedman has defined the controllability and observability as the factor for the assessment of testability of a component.

From the study of Se Do Sohn et. al. [45] in digital safety has shown fault tree based analysis to analyze the testability. They had back tracked the failure output to the input event. So, this study showed that the fault tree analysis is also applicable approach to analyze the testability of a program. From the study of Magiel Bruntink [10], the class testability can be assessed by analyzing two categories of source code level factors – test case generation factors and test case construction factors.
In conclusion, we can derive an undesired state of the program in assertion of the test suites. If we consider on the freedman’s concept for testable component and Magiel Bruntink [10] concept, we can derive different undesired state of the program in the process of generating the test suites. For example, difficult to specify test inputs, difficult to generate test cases, difficult to interpret test outputs, difficult to setup test environment, difficult to determine internal states of the program.

We have proposed a systematic approach for the testability assessment based on the fault tree analysis. In the assessment, a high level undesired event/state of the program is identified first. Then, an iterative process of breaking down that event/state to the directly measureable and primary root cause factors is carried out. The process is depicted as in the following figure:

**Figure 6.4: Proposed testability assessment framework**

**Assessment Process:**
To identify the affecting factors, we have to start from the high level undesired level. Then, all immediate factors are listed in next level. Each factor is evaluated based on the cause and effect analysis so that top level state must be the effect of lower level factor as a cause. Then a gate function is identified based on the relation between the factors. If two or more factors simultaneously result the effect they are linked with AND gate.
and if they affect individually, then they are linked with OR gate and so on. After that, the necessary and sufficient condition is applied to reduce the factors when possible. Detail process is described below.

Step 1: start from the high level (level 0) undesired state
Step 2: List all possible factors that will cause high level (level 0) undesired state
Step 3: Review the gate (Gate1) event for the factors and identify condition or logic for cause and effect.
Step 4: keep looking back to ensure factors are not jumped ahead and repeated as well.
Step 5: repeat for process for next gate (Gate 2 and so on) from the previous level’s (level 1) undesired state.

At each Gate function:

- Check for immediate, necessary and sufficient conditions to sort the factors affecting testability.
  
  Step 1: Find all possible immediate factors to cause the undesired state. Immediate – all possible factors from the different domains like input, output, test environment etc.
  
  Step 2: structure the tree with necessary and sufficient conditions to cause the upper level undesired event.
    
    Necessary - include the factors that are actually necessary
    Sufficient – include the factors that are minimum necessary
  
  Step 3: repeat the process for next gate function

- Check for primary, secondary and undeveloped factors
  
  Step 1: For each sufficient factor, determine whether it is directly measureable or needs to be break down into another lower level to measure or it is completely undeveloped.
  
  - Assign circle symbol for primary factor, rectangle for secondary and diamond for undeveloped factor.

After completing the construction of the tree, Fussel-Vesely algorithm is applied to find out the cut sets of factors affecting the overall top level undesired event.

- traverse the tree from the top level
- increase the order of factors when encounter a AND gate
- decrease the order of factors when encounter an OR gate
7 CASE STUDY

This chapter shows a case study for the validation of the proposed framework. The framework is applied to find the difficulties in the program to locate the faults. Four failure cases are selected for a program written in Mathematica (a master thesis project in BTH). The cases are chosen based on the time taken to debug the program. Later, the framework is applied to analyze the program and debugged the program again. The time taken for debugging after intervention of the analysis result are taken and made the direct comparison to conclude the applicability of the framework.

7.1 Case Study A: Case Study Setup

Problem statement:
- Identifying the Location of bug in the program when a failure is seen.

Objective and goal:
- Identifying the factors causing the failure in the program.

Hypothesis

Null hypothesis

H0: There will be same debug time required for the program after the intervention of proposed framework in debugging.

H1: There will be less debug time required for the program after the intervention of proposed framework in debugging.

7.1.1 Motivation and Alternatives

The proposed framework has in its primitive stage i.e. it is only a conceptual framework. This framework is intended to be used as the self assessment of testability for test engineers as presented in the thesis proposal. So, the intervention introduced in the hypothesis is target to the novice test engineers. To test this hypothesis, we can select experiments in real industrial environment, simulation with dummy program and case studies. We excluded the former two methods due to following reasons:

- The framework is in its conceptual stage which may be liable to change. In such situation the industrial experiment cannot be preferable due to time and cost factors.
- Industries and participants may not be very keen as it has not shown even a pilot studies about the validation.
- Even though, the framework is based on the high quality articles and systematic studies, the implementation preparation may not sufficient due to time factor required for such experiments.

We have chosen the case studies. The case study includes a master thesis program in electrical engineering at BTH in 2011, September. The case study is chosen based on following reasons:

- The case study with participants of the master thesis satisfies the target population as novice testers.
- The program written for the simulation needs the debugging and it will provide the real time environment.
- The case study will not be vulnerable for the changes made in framework as it can be used as pilot validation.

7.1.2 Methodology

We used the naturalistic observation research methodology for the case study. In this methodology, study is done in the natural habitat i.e. the natural environment and there will not any manipulation of the environment by observer nor creates such environment. The method used for the observation is unobtrusive observation where
observer does not participate in the process but observe the behavior and records data. We did not participated in the debugging process of the program at all.

7.2 Program Description
Detection of probability of cognitive radio network over correlated Nakagami-M fading channel was a master thesis in BTH. Thesis author is a master student in Telecommunication program. The thesis work involved the simulation of the cognitive radio network in Mathematica program. The author of thesis has designed and developed the simulation program.

![Diagram of PD program](image)

Figure 7.1: System description of PD program

Program has been designed as shown in above figure. There was a main function called PD. The main function takes a single input SNR (signal to noise ratio) values and calculates the detection probability for the network and displays the result on console. For the calculation, main function calls the sub routines integral and partfracof (partial fraction coefficient).

During the debugging session of the program, we observed the different failure states for the program and selected four failure cases for the study.

7.3 Execution
The execution of the case study was conduction purely in the environment of the thesis work. We divided the participants in two groups. In the first group, the developer of was considered. In this group, we did not use the intervention of the framework. Instead, we observed the debugging process and obtained the four failure cases during the development of the program and recorded the time taken to locate the fault.

The identified failure cases were:
- Case 1: output of the program is “indeterminate”
- Case 2: output out of range (due to wrong variable passed in a module.)
- Case 3: output out of range (due to wrong expression)
- Case 4: high execution time (due to function call from wrong location)

In the second group, other three colleagues were chosen. The three colleagues debugged the program for the failure cases indentified. Before starting, the participants were participated to learn the concept of the proposed framework and constructed the testability tree for three different testability forms and hence identified the factors that could make difficulty in locating the fault. After this intervention, the normal debugging process was carried out and noted the time taken for debugging in the same way.
7.4 Result
From the observation without intervention, we recorded the following debugged time:

Table 7.1: Experiment result for participant A

<table>
<thead>
<tr>
<th>SN</th>
<th>Participant</th>
<th>Failure</th>
<th>Debug time</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>indeterminate</td>
<td>7days</td>
<td>developer</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>Response time &gt;20 minutes</td>
<td>14days</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>Output &lt;0</td>
<td>7days</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>Output &gt;1</td>
<td>3days</td>
<td></td>
</tr>
</tbody>
</table>

From the intervention we recorded the following result.
Testability tree for “Difficult to specify inputs”

Figure 7.2: Testability tree for difficult to specify inputs
The cut set identified {No of internal inputs increased},{No coding standard}
For Testability tree for “Difficult to interpret output”

Figure 7.3: Testability tree for difficult to interpret output

The cut sets identified

{No coding standard followed}, {output can be defined only in the range, same output for different inputs may exits},{ output dependent to internal outputs, output can be defined only in the range}
Figure 7.4: Testability tree for difficult to trace internal variables

For Testability tree for “Difficult to trace internal variables”
{No coding standard followed},{ No of variables are high},{ No of loop count is high},{ coupling between modules is high}

After intervention, the debugging times for the program recorded were:

Table 7:2: Experiment result for participant B

<table>
<thead>
<tr>
<th>SN</th>
<th>Participant</th>
<th>Failure</th>
<th>Debug time</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B</td>
<td>indeterminate</td>
<td>7days</td>
<td>Not a developer</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>Response time &gt;20 minutes</td>
<td>9days</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td>Output &lt;0</td>
<td>4days</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>Output &gt;1</td>
<td>1days</td>
<td></td>
</tr>
</tbody>
</table>
Table 7:3: Experiment result for participant C

<table>
<thead>
<tr>
<th>SN</th>
<th>Participant</th>
<th>Failure</th>
<th>Debug time</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C</td>
<td>indeterminate</td>
<td>3days</td>
<td>Not a developer</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>Response time &gt;20 minutes</td>
<td>6days</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>Output &lt;0</td>
<td>3days</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>C</td>
<td>Output &gt;1</td>
<td>1days</td>
<td></td>
</tr>
</tbody>
</table>

Table 7:4: Experiment result for participant D

<table>
<thead>
<tr>
<th>SN</th>
<th>Participant</th>
<th>Failure</th>
<th>Debug time</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>D</td>
<td>indeterminate</td>
<td>4days</td>
<td>Not a developer</td>
</tr>
<tr>
<td>2</td>
<td>D</td>
<td>Response time &gt;20 minutes</td>
<td>8days</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>D</td>
<td>Output &lt;0</td>
<td>2days</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>D</td>
<td>Output &gt;1</td>
<td>1days</td>
<td></td>
</tr>
</tbody>
</table>

From the direct comparison between the results of two groups, we conclude that after the intervention the debug time taken is reduced for the same failure. Hence the result has supported the alternative hypothesis and rejected the null hypothesis.

7.5 Limitation

The case study has following limitation:

- **Time Consuming**
  
  We have taken the case study for the master thesis. The case study involved the participants who are responsible to develop the program and debug. So, we have to coordinate with their progress. Further, the method used for the case study is naturalistic observation. We cannot influence the process as well. The impact of this context prolonged the data collection time.

- **Poor Representativeness**
  
  The participants are students in the master thesis. So, the representativeness of target group is poor as compared to the participants from the industry. Although, the participants are student, they represent the target group as they have developed program that shows the knowledge of requirement, design and development of the software project. Moreover, they are native in testing field which is the strong point for us as we are focused on the native testers as the target group.

- **Cannot Generalize beyond those studies**
  
  The case study result is based on the four failure cases that took more time to fix for the developer of the program. The program is not the developed and debugged in the
software development environment. But, the work resembles the real environment in a sense that it has fixed requirement for which a fault in the program can mislead to the result of their thesis. So, it is necessary to be sure that the faults are not in the program. Even though, the results cannot be generalized to the software development environment.

7.6 Case Study B: Dependency Example

In the second case study, we studied the dependency problem that can be present in the test cases inside a test suite. If a test case requires the execution of another operation before, then their exits dependency. And if this dependency exists in test cases there will be a chance of reporting false bug when the previous case fails. The following test code shows the problem.

```java
@Test(priority = 0)
public void ReadCurrentYear()
{
    Assert.assertEquals(2010, PostMain.CurrentYear);
}

//@Test(dependsOnMethods = {"ReadCurrentYear")
@Test(priority = 1)
public void UpdateCurrentYear()
{
    PostMain.UpdateCurrentYear();
    Assert.assertEquals(2012, PostMain.CurrentYear);
}

//@Test(dependsOnMethods = {"UpdateCurrentYear")
@Test(priority = 2)
public void GotMarried()
{
    Assert.assertEquals("Person X", PostMain.GotMarried());
}
```

In the code, there is a “CurrentYear” variable which is global. There are three methods to access the global CurrentYear to read, write and to do some operations based on the value of that variable.

Test scenario is to check for default value, update and confirm it for desired value and check for a person is married on that year.

From the experiment, we found that the order of test cases is random as each test case in supposed to be independent.

To maintain the order we used the priority for the test cases as required.

Then the result we got was leading to a false error which is introduced due to the error in test case not in the actual program in third method.

So, the testability analysis will help to find what functions are dependent on which and the order of the execution required. Thus, what factor is important while writing test cases in the working environment? For example, here the isolation of the third method with second method is the problem as it is dependent on global CurrentYear variable.
8 DISCUSSION

In context of answering the research question 1 and 2 the different testability factors as well as testability measurements were identified. The testability factors that we have summarized form the systematic literature review is shown figure... The mentioned factors are the important factors that affect the software testability. These are the factors that need to be considered while measuring the software testability. These factors are considered from the runtime systems to component based system, from object oriented system to other structured programs. These factors affect the different software development life cycle stages from requirement to design, coding etc.

![Diagram](attachment:image.png)

Figure 8.1: Software Testability Factors

![Diagram](attachment:image.png)

Figure 8.2: List of Factors affecting software testability
Figure 8.2 Shows the testability measurements identified in conducting the systematic literature review.

![Diagram of Testability Metrics]

**Figure 8.3: List of Testability metrics**

Above are some of the testability metrics used in the software field. We have found that software testability methods are mainly categorized according to the cost analysis and sensitivity analysis. With the distinction between testability measurement methods fall under software structural or object oriented methods, the ultimate goal will be test resources prediction and test effectiveness and finally helps in project decision making.

The section below will discuss how the proposed framework of testability presented in the answer of RQ3 will be help to the testers while writing the test suites.

Test suite is a collection of test cases that are grouped together for specific purpose. The test cases are grouped on the basis of criterion of the testing for example: based on features, function, test coverage criteria etc. Further, the test suite also describes the prerequisites to execute the grouped test cases. So, a test suite explain the purpose of the test, test environment setup and test cases, along with referring to the requirements related to that test suite. A test suite will pass if all the test cases included in the suite pass the test.

A test case, on the other hand, is a set of inputs, conditions and expected output. Each test case in a suite will have probability of detection the fault, if there exists in the program under test.

While analyzing the test suite, we can clearly see that there are different domains that should be considered. The following picture will show more clearly.
Figure 8.4: Domains of a test suite

A test suite and test cases can be written if we can specify the inputs for the program under test, its expected outcome and requirements for the test setup. A decision can be made whether test case is passed or failed, when we can compare the expected output with the actual output referred to the requirement. So, difficulty or complexity to specify the variable for these domains will degrade the testability of the program. The existing testability analysis methods scrutinize the difficulties or complexity in the program related to those domains in some ways. So, from the existing testability analysis methods, we can assess the factors that could affect to testability and transform into one or more domains of test suite explained.

In the proposed framework (answer to RQ3), we have used the different perspective into a single framework.

- Testability form i.e. high level undesired state of the program. More clearly, the state of program being not testable.
- Testability tree based on fault tree.
- Testability affecting factors based on the testability analysis.

The output of the framework is the minimal cut sets of testability affecting factors that are responsible for making test suite generation difficult. The information gathered regarding to these factors will help to decide where the tester should focus more. The proposed framework has following characteristics:

- It is a graphical method which will be easy to understand and use.
- Based on the problem (can be assumed or facing).
It is well structured and can accumulate the concepts of other testability analysis method into it.

It is a flexible framework which can be used in the testability assessment of the full or partial system of concern.

It can be used early in the software development process as well as just beginning of the test plan.

It is qualitative approach, but can be extended to quantitative assessment based on the FTA for example reliability, effort estimation etc.

Co-relation between the factors can also be determined and priority of the factors also identified.
9 **Validity and Threats**

The threats to validity affect the accuracy of the result under investigation. The results of the research study in are affected by the different factors. The different methods of research may have the different threats to validity. A systematic literature and case study experiments have the internal, external, construct and conclusion validity threats [76].

9.1 **Internal Validity**

We have considered the following internal validity threats for the systematic literature review:

**Publication Bias**: the papers collected for systematic review may only have the results which have affirmative conclusions because the publication may be positive only for such papers. Such kind of bias cannot be minimized completely. But, we have referred more than one paper for same research topic wherever possible and look into multiple e-databases as well. It will reduce the extent of negative impacts and use of trusted databases will also be help as the papers are peer-reviewed.

**Selection bias**: the manual paper selection process may favor to those papers of individual’s desirability. To avoid this bias we have used pre-defined inclusion and exclusion criteria in review protocol for systematic review. Selection process is computer assisted whenever possible.

**False inference**: It is a threat to internal validity in which incorrect inferences are derived from gathered data. To mitigate this, we have used a pre-defined systematic review protocol and has been used the multiple viewpoints. Besides that the original viewpoint has been presented whenever required. Similarly, peer review has been used to make agreement between the participants to reach the final conclusion.

**Learning effect**: It is also a threat to internal validity in experiment. A team writing test cases for a program in untreated approach can learn where to focus for better result. And the same team when writes the test cases in treated approach might use that knowledge. Then about the experiment result, we cannot assure that the result reached in second scenario is due to the knowledge gained from first scenario or actually from the only second scenario. It can be termed as learning effect and to avoid this we will conduct the experiment with two team dealing with two similar programs for untreated and swap the program for treated scenario.

9.2 **External Validity**

The external validity identified for this study is related to the threats to repeatable of the process practiced during the study [73]. If the processes do not comply with the repeatability, then the result could be out of the scope for each repeated process and the result cannot be generalized. To address the threats, we developed the protocol for the systematic review prior to start the processes and followed that protocol. The key papers, kappa analysis for the mutual agreement level are the measures used to control the external validity.

9.3 **Construct Validity**

The construct validity threat is related to the relation between the theories and surveillances [73]. In our research, the threat is related to generalize the proposed framework with the concepts used in the formation of framework. The systematic review of the factors and its breakdown from high level of abstraction to measureable level abstraction has been performed. The framework is based on the reviewed facts and is hypothetically tested in the validation work.
9.4 Conclusion Validity

Conclusion validity is known as the reliability of the study in research. It concerns with the reliability of the result leading with correct conclusion [73]. To achieve the reliability of study, we have defined the review protocol based on the guidelines given by Kitchenham [73]. The key papers are used to find the coverage of search strings, pilot search and selection, predefined protocol update rules, mutual agreement level check between the authors by kappa analysis and validation of deliverables by the supervisor.
10 **CONCLUSION**

This section summarizes the work done during thesis. The major contribution and limitation is presented in the section.

This thesis presented the systematic literature studies in the field of software testability. The primary focus of the study was to identify the testability factors presented in the current research domain and their measurements.

10.1 **Contribution**

The main contribution of this thesis is in the field of testability assessment for test suite generation. We have conducted a systematic review in this field. The different factors testability and measurement for these factors are identified. Overall contribution is listed as follows:

- **Systematic Literature Review**
  - A complete step by step development of the systematic review process is described. It will help to other researchers as a reference for undertaking SLR.
  - Identification of key papers related to the testability analysis in software engineering domain
  - Identification of testability factors and measurement in the current domain of software engineering.
  - Identification and presentation of different concepts about the software testability in the current software engineering domain.

- A proposed software testability framework to help the self-assessment for test engineers to identify software testability factors.
  - Identification of high level testability form regarding to test suite
  - Structure and well defined assessment process for finding factors from high level to lower measurable level.
  - Integration of FTA concept in the testability assessment, which provides the graphical and Boolean algebraic logic in qualitative approach.

10.2 **Limitation**

During the course of planning and conducting the systematic review and validating the framework with the target population, we have some limitation.

- Systematic literature review includes the only six e-databases. There is chance of missing some important articles related to the field of study which are published from other source.
- The exclusion of articles based on full text availability also is an important issue.
- The validation was conducted based on case study in master thesis. So, the validation is not based on the industrial context yet. The reason for choosing this validation is, we have to find some evidences to take this for experiment or survey in industrial field. So, this validation is, somehow, regarded as the pilot study.
  
  Since, there is few number of participants whose activities are observed and only four cases are included, the generalization may be limited. But, we have addressed the target population as the participants.
11 **RECOMMENDATION AND FUTURE WORK**

The framework has been developed based on the systematic literature review. The framework has shown the improvement of performance in the case study as well. But, due to poor representation, the validation cannot be generalized. So, there is a need of validation from the real industrial area. Therefore, the recommended future works are:

- The validation from the industrial environment with testing practitioner will definitely help to develop the framework and generalize it.
- The framework has developed based on the FTA and used the qualitative analysis of the factors. The cut sets are the result of the qualitative analysis. The FTA method can be used for the quantitative analysis. Hence, the identified cut sets can be further used for quantitative analysis and thus estimation of reliability, effort can be done for the testing process.
REFERENCES


ISO 9126, International Organization for standardization

P. Edwards “The Effect of certain modular design principles on software testability”. *CAN SIGPLAN NOTICES*, Volume 10(6), Page(s) : 401-410, April 1975


Miller, David B., Roles of naturalistic observation in comparative psychology, American Psychologist, Vol 32(3), Mar 1977, 211-219


DOI: [http://www.ei.org/compendex](http://www.ei.org/compendex)

DOI: [http://www.itc.nl/Pub/Home/library/Library-general-information/more-info-databases/SpringerLink_journals_info.html](http://www.itc.nl/Pub/Home/library/Library-general-information/more-info-databases/SpringerLink_journals_info.html)
## APPENDIX-I

Documents used for pilot, inclusion/ exclusion criteria and kappa analysis.

<table>
<thead>
<tr>
<th>S.N.</th>
<th>Article Name</th>
<th>Full Text</th>
<th>S.N.</th>
<th>Article Name</th>
<th>Full Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Event-based architectures: designing for testability and maintainability</td>
<td>NO</td>
<td>41</td>
<td>Testing of a highly reconfigurable processor core for dependable data streaming applications</td>
<td>YES</td>
</tr>
<tr>
<td>2</td>
<td>Design for testability method to avoid error masking of software-based self-test for processors</td>
<td>NO</td>
<td>42</td>
<td>Software design of grounding grid corrosion diagnosis based on hierarchical simplification of network topology</td>
<td>NO</td>
</tr>
<tr>
<td>3</td>
<td>A study on design for testability in component-based embedded software</td>
<td>YES</td>
<td>43</td>
<td>Automating HIL test setup and execution</td>
<td>YES</td>
</tr>
<tr>
<td>4</td>
<td>Survey of software design for testability</td>
<td>NO</td>
<td>44</td>
<td>Software self-testing of a symmetric cipher with error detection capability</td>
<td>YES</td>
</tr>
<tr>
<td>5</td>
<td>A metadata model based on coupling testing information to increase testability of component</td>
<td>NO</td>
<td>45</td>
<td>Proceedings IEEE AUTOTESTCON 2008</td>
<td>NO</td>
</tr>
<tr>
<td>6</td>
<td>Evaluating evolutionary testability for structure-oriented testing with software measurements</td>
<td>YES</td>
<td>46</td>
<td>Search based software testing of object-oriented containers</td>
<td>YES</td>
</tr>
<tr>
<td>7</td>
<td>Testability analysis method for hardware and software based on assertion libraries</td>
<td>YES</td>
<td>47</td>
<td>SoC Software Components Diagnosis Technology</td>
<td>YES</td>
</tr>
<tr>
<td>8</td>
<td>An analysis technique to increase testability of object-oriented components</td>
<td>YES</td>
<td>48</td>
<td>Teaching digital test with BIST analyzer</td>
<td>YES</td>
</tr>
<tr>
<td>9</td>
<td>Modelling variability and testability interaction in software product line engineering</td>
<td>YES</td>
<td>49</td>
<td>Guidelines for modelling the Arinc 653 software architecture</td>
<td>NO</td>
</tr>
<tr>
<td>10</td>
<td>Design and analysis of complex system testability based on hybrid diagnostic</td>
<td>NO</td>
<td>50</td>
<td>Software environment for</td>
<td>NO</td>
</tr>
<tr>
<td>S.N.</td>
<td>Article Name</td>
<td>Full Text</td>
<td>S.N.</td>
<td>Article Name</td>
<td>Full Text</td>
</tr>
<tr>
<td>------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------------------------</td>
<td>------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>11</td>
<td>Testability analysis based on the identification of testable blocks with predefined properties</td>
<td>YES</td>
<td>51</td>
<td>Scan BIST with biased scan test signals</td>
<td>NO</td>
</tr>
<tr>
<td>12</td>
<td>Testability of dynamic real-time systems: an empirical study of constrained execution environment implications</td>
<td>NO</td>
<td>52</td>
<td>A deterministic methodology for identifying functionally untestable path-delay faults in microprocessor cores</td>
<td>NO</td>
</tr>
<tr>
<td>13</td>
<td>&quot;Plug &amp; Test&quot; at system level via testable TLM primitives</td>
<td>YES</td>
<td>53</td>
<td>Component-based, run-time flight software modification</td>
<td>NO</td>
</tr>
<tr>
<td>14</td>
<td>Open problems in testability transformation</td>
<td>NO</td>
<td>54</td>
<td>A scalable method for testing real-time systems</td>
<td>YES</td>
</tr>
<tr>
<td>15</td>
<td>Testability of Web service</td>
<td>NO</td>
<td>55</td>
<td>A secure information flow architecture for Web services</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Research on key technology in realizing assistant decision-making system of design for testability of equipment</td>
<td>NO</td>
<td>56</td>
<td>Connector-driven gradual and dynamic software assembly evolution</td>
<td>YES</td>
</tr>
<tr>
<td>17</td>
<td>Estimation of test code changes using historical release data</td>
<td>YES</td>
<td>57</td>
<td>HW/SW co-simulation platforms for VLSI design</td>
<td>YES</td>
</tr>
<tr>
<td>18</td>
<td>Building testable components—a systematic approach and its experimental study</td>
<td>NO</td>
<td>58</td>
<td>GUI Testing Made Easy</td>
<td>YES</td>
</tr>
<tr>
<td>19</td>
<td>Test by contract for UML-based software component testing</td>
<td>YES</td>
<td>59</td>
<td>Interactive fault finding on printed circuits, using boundary scan test methods</td>
<td>NO</td>
</tr>
<tr>
<td>20</td>
<td>A service oriented architecture complexity metric, based on statistical hypothesis testing</td>
<td>NO</td>
<td>60</td>
<td>A modular system architecture for autonomous robots based on blackboard and publish-subscribe mechanisms</td>
<td>YES</td>
</tr>
<tr>
<td>S.N.</td>
<td>Article Name</td>
<td>Full Text</td>
<td>S.N.</td>
<td>Article Name</td>
<td>Full Text</td>
</tr>
<tr>
<td>------</td>
<td>------------------------------------------------------------------------------</td>
<td>-----------</td>
<td>------</td>
<td>------------------------------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>21</td>
<td>Comparing aspects with conventional techniques for increasing testability</td>
<td>YES</td>
<td>61</td>
<td>Design of multiple scan tree with extended compatibilities</td>
<td>NO</td>
</tr>
<tr>
<td>22</td>
<td>Research on deciding testability figures of complicated equipment</td>
<td>NO</td>
<td>62</td>
<td>Design and Implementation of a single-chip ARM-based USB interface JTAG emulator</td>
<td>YES</td>
</tr>
<tr>
<td>23</td>
<td>Improved test efficiency in cores-based system-on-chips using ModelSim verification tool</td>
<td>NO</td>
<td>63</td>
<td>DFT, ATE drive yield improvement</td>
<td>NO</td>
</tr>
<tr>
<td>24</td>
<td>On extracting tests from a testable model in the context of domain engineering</td>
<td>YES</td>
<td>64</td>
<td>A reconfigurable scan architecture with weighted scan-enable signals for deterministic BIST</td>
<td>YES</td>
</tr>
<tr>
<td>26</td>
<td>Testing consequences of grime buildup in object oriented design patterns</td>
<td>NO</td>
<td>65</td>
<td>Software BIST capabilities of a symmetric cipher</td>
<td>YES</td>
</tr>
<tr>
<td>27</td>
<td>Testing consequences of grime buildup in object oriented design patterns</td>
<td>NO</td>
<td>66</td>
<td>Practical challenges in logic BIST implementation - case studies</td>
<td>YES</td>
</tr>
<tr>
<td>28</td>
<td>Model based testing of system requirements using UML use case models</td>
<td></td>
<td>67</td>
<td>E-RACE, a hardware-assisted approach to lockset-based data race detection for embedded products</td>
<td>YES</td>
</tr>
<tr>
<td>29</td>
<td>A metadata configuration model for component-based software integration testing</td>
<td>NO</td>
<td>68</td>
<td>Research on system-level test of VLSI based on boundary scan technology</td>
<td>NO</td>
</tr>
<tr>
<td>30</td>
<td>SoC symbolic simulation: a case study on delay fault testing</td>
<td>NO</td>
<td>69</td>
<td>Exploring the concept of systems theoretic stability as a starting point for a unified theory on software engineering</td>
<td>YES</td>
</tr>
<tr>
<td>31</td>
<td>An empirical study into use of dependency injection in Java</td>
<td>NO</td>
<td>70</td>
<td>Data exchange</td>
<td>NO</td>
</tr>
<tr>
<td>32</td>
<td>The research on the application of UML</td>
<td>NO</td>
<td>71</td>
<td>The embedded</td>
<td>NO</td>
</tr>
<tr>
<td>S.N.</td>
<td>Article Name</td>
<td>Full Text</td>
<td>S.N.</td>
<td>Article Name</td>
<td>Full Text</td>
</tr>
<tr>
<td>------</td>
<td>------------------------------------------------------------------------------</td>
<td>-----------------</td>
<td>------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>33</td>
<td>Bridging best traditional SWD practices with XP to improve the quality of XP projects</td>
<td>YES</td>
<td>72</td>
<td>New techniques for accelerating small delay ATPG and generating compact test sets</td>
<td>YES</td>
</tr>
<tr>
<td>34</td>
<td>Architecture support for runtime integration and verification of component-based systems of systems</td>
<td>NO</td>
<td>73</td>
<td>Short circuit [vehicle electronics]</td>
<td>NO</td>
</tr>
<tr>
<td>35</td>
<td>A comparative study of maintainability of Web applications on J2EE, .NET and Ruby on Rails</td>
<td>YES</td>
<td>74</td>
<td>SitCom: virtual smart-room environment for multi-modal perceptual systems</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>Systematic software-based self-test for pipelined processors</td>
<td>YES</td>
<td>75</td>
<td>Digital RF processor (DRP™) for wireless transmitters</td>
<td>YES</td>
</tr>
<tr>
<td>37</td>
<td>An approach to addressing entity model variability within software product lines</td>
<td>YES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>Why developers insert security vulnerabilities into their code</td>
<td>YES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>DCScan: a power-aware scan testing architecture</td>
<td>YES</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX-II

Quality Assessment