Final Thesis

Systematic Review on Automated Testing
Types, Effort and ROI

Imran Muneer

LIU-IDA/LITH-EX-A--14/004--SE

2014-01-28

Supervisor: Johan Åberg
Examiner: Kristian Sandahl
Abstract
Software organizations always want to build software by minimizing their resources to reduce overall cost and by maintaining high quality to produce reliable software. Software testing helps us to achieve these goals in this regard. Software testing can be manual or automated. Manual testing is a very expensive activity. It takes much time to write test cases and run them one by one. It can be error-prone due to much involvement of human throughout the process. Automated testing reduces the testing time which results in reduction of overall software cost as well as it provides other benefits i.e. early time to market, improved quality. Organizations are willing to invest in test automation. Before investment, they want to know the expected cost and benefits for AST. Effort is the main factor, which increase the cost of testing.

In this thesis, a systematic review have been conducted which identifies and summarizes all the retrieved research concerning the automated testing types, effort estimation and return on investment (ROI) / cost-benefit analysis for automated testing. To conduct the systematic review, the author has developed a comprehensive plan which follows the procedure presented in [15]. This plan provides guidance to identify relevant research articles of a defined period. After the identification of research articles, it collects, evaluates and then interprets all the retrieved data about automated testing types, effort estimation and ROI. The results have been presented in statistical and descriptive form which provides different aspects of the data.

The statistical results have been presented with the help of tables and graphs which show different aspects of data i.e. any gaps in research work of automated testing, number of articles for each testing type. The answers of the questions have been presented in descriptive form. The descriptive results show 22 automated testing types, 17 Industrial case studies out of 60 studies, benefits of automated testing and effort estimation models. The discussion part highlighted some important facts about the retrieved data and provides practical implications for conducting systematic reviews. Finally it is concluded that systematic reviews are good means of finding and analyzing research data about a topic, phenomena and area of interest. It also provides support to researchers for conducting and investigating more research.
Acknowledgments

First of all, I am thankful to Almighty Allah who gave me the wisdom and ability to complete my master study. My every word of praise is ultimately for him.

I would like to express my profound thankfulness to my thesis supervisor Prof. Johan Åberg and thesis examiner Prof. Kristian Sandahl for providing me the opportunity to work with them and for their valuable guidance, suggestions & support throughout the study period, especially during my thesis work.

I would also like to thank all my teachers and faculty of IDA department for their valuable knowledge and assistance during the study of courses at Linköping University.

I am thankful to all my friends especially Nasir Zaman, Muhammad Hussain and class fellows at Linköping University for their constructive and moral support through discussions and suggestions during my study and thesis work.

I am also thankful to my family especially my parents and brothers for their eternal support and encouragement throughout my life.

I again thank to all of you for their support and encouragement. I was unable to reach at this stage without all of you.

Imran Muneer
**Abbreviations & Glossary of Terms**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AST</td>
<td>Automated Software Testing</td>
</tr>
<tr>
<td>STL</td>
<td>Software Testing Lifecycle</td>
</tr>
<tr>
<td>CBSE</td>
<td>Component Based Software Engineering</td>
</tr>
<tr>
<td>SUT</td>
<td>Software Under Test</td>
</tr>
<tr>
<td>ROI</td>
<td>Return on Investment</td>
</tr>
<tr>
<td>ACO</td>
<td>Ant Colony Optimization</td>
</tr>
<tr>
<td>AVM</td>
<td>Alternating Variable Method</td>
</tr>
<tr>
<td>TTCN-3</td>
<td>Testing and Test Control Notation version 3</td>
</tr>
<tr>
<td>TSE</td>
<td>Test Set Editor</td>
</tr>
<tr>
<td>BERT</td>
<td>BEhavioral Regression Testing</td>
</tr>
<tr>
<td>AUT</td>
<td>Application Under Test</td>
</tr>
<tr>
<td>DART</td>
<td>Daily Automated Regression Tester</td>
</tr>
<tr>
<td>DART</td>
<td>Distributed Automated Regression Testing</td>
</tr>
<tr>
<td>JSART</td>
<td>JAVASCRIPT Assertion-based Regression Testing</td>
</tr>
<tr>
<td>OCL</td>
<td>Object Constraint Language</td>
</tr>
<tr>
<td>DART</td>
<td>Directed Automated Random Testing</td>
</tr>
<tr>
<td>AAS</td>
<td>Actual Action Sequence</td>
</tr>
<tr>
<td>EAS</td>
<td>Expected Action Sequence</td>
</tr>
<tr>
<td>FIT</td>
<td>Framework for Integrated Tests</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>TSL</td>
<td>Test Specification Language</td>
</tr>
<tr>
<td>IDATG</td>
<td>Integrating Design and Test-Case Generation</td>
</tr>
<tr>
<td>TESLA</td>
<td>Test Specification Language</td>
</tr>
<tr>
<td>CAST</td>
<td>Computer-Aided Specification and Testing</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithm</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>ROHC</td>
<td>Robust Header Compression</td>
</tr>
<tr>
<td>MAIT</td>
<td>Model for Automated Interoperability Test</td>
</tr>
<tr>
<td>JAT</td>
<td>Jade Agent Testing Framework</td>
</tr>
<tr>
<td>MAS</td>
<td>Multi-Agent System</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>RPB</td>
<td>Record and PlayBack</td>
</tr>
<tr>
<td>TEF</td>
<td>Technical Environment Factor</td>
</tr>
<tr>
<td>AUCP</td>
<td>Adjusted Use Case Point</td>
</tr>
<tr>
<td>UUCW</td>
<td>Unadjusted Use Case Weight</td>
</tr>
<tr>
<td>UAW</td>
<td>Unadjusted Actor Weight</td>
</tr>
</tbody>
</table>
List of Tables

Table 2.1 Quality Item Checklist
Table 2.2: Extraction of Data Items
Table 2.3: Data Synthesis form
Table 3.1: selection of articles
Table 3.2 Selected articles with titles and years
Table 3.3: Number of articles for each testing type
Table 3.4 Industrial Case Studies for each testing type
Table 3.5: ROI calculation
Table 3.6: Benefits of automated Testing
List of Figures & Graphs

Fig. 1: Automated Testing Process

Graph 3.1 Number of selected articles over the period 1998-2013
Graph 3.2 Research methodology used in selected Articles
Graph 3.3: Number of Selected Articles per Study Environment
Graph 3.4: Number of Selected Articles per Study Environment over the period 1998-2013
Graph 3.5 Number of articles for each testing type
Graph 3.6: Number of articles over the period 1998-2013 for each Testing Type
Table of Contents
Abstract........................................................................................................................................................ iii
Acknowledgments........................................................................................................................................ iv
Abbreviations & Glossary of Terms .............................................................................................................. v
List of Tables ................................................................................................................................................ vi
List of Figures & Graphs .............................................................................................................................. vii
1. BACKGROUND....................................................................................................................................... 1
   1.1. TESTING TYPES .............................................................................................................................. 1
       1.1.1. Black-Box Testing...................................................................................................................... 1
       1.1.2. White-Box Testing................................................................................................................... 2
   1.2. AUTOMATED TESTING PROCESS...................................................................................................3
       1.2.1. Test Environment Setup ........................................................................................................ 4
       1.2.2. Test Development / Creation.................................................................................................. 4
       1.2.3. Test Execution and Results Evaluation..................................................................................... 5
   1.3. PURPOSE ....................................................................................................................................... 7
   1.4. RESEARCH QUESTIONS ..................................................................................................................8
   1.5. THESIS STRUCTURE .......................................................................................................................8
2. SYSTEMATIC REVIEW METHOD.............................................................................................................9
   2.1. SEARCH STRATEGY ........................................................................................................................9
   2.2. STUDY SELECTION .........................................................................................................................9
       2.2.1. Inclusion Criteria...................................................................................................................... 10
       2.2.2. Exclusion Criteria .................................................................................................................. 10
   2.3 STUDY QUALITY ASSESSMENT ....................................................................................................10
   2.4. DATA EXTRACTION......................................................................................................................11
   2.5. DATA SYNTHESIS .........................................................................................................................12
3. RESULTS AND ANALYSIS......................................................................................................................14
   3.1. SELECTION OF STUDIES .............................................................................................................14
   3.2. ANALYSIS OF SELECTED ARTICLES ...........................................................................................17
   3.3. AUTOMATED TESTING TYPES ......................................................................................................21
   3.4. INDUSTRIAL CASE STUDIES .......................................................................................................35
   3.5. RETURN ON INVESTMENT (ROI) FOR TEST AUTOMATION .........................................................36
   3.6. BENEFITS OF AUTOMATED TESTING ..........................................................................................37
1. BACKGROUND
Software industry has been moving towards building larger software that are more complex in nature. Recent advancements in technology have created more challenges for Software organizations to develop these large and complex softwares. Software organizations always want to build these software by minimizing their resources and time to reduce overall cost and by maintaining high quality to produce reliable software. Software testing helps us to achieve these goals in this regard. According to [1], “software testing is a process of ensuring that a certain piece of software item fulfills its requirements”

There is always a market demand to buy software that are error free, provide more functionality than expected and with low price. With all these demands, a competition always continues in the software industry. A report published by NIST describing “Reducing the cost of software development and improving software quality are important objectives of the U.S. software industry”. This report also describes that “Software errors cost the U.S. economy an estimated $59.5 billion annually, of which they estimate one-third, could be eliminated by improved testing” [2]. But how can Organizations improve their software testing process to reduce software development cost and to achieve better software quality? This can be achieved by implementing a good testing strategy.

1.1. TESTING TYPES
There are two main categories of software testing.

1.1.1. Black-Box Testing
Black-box testing is “a strategy in which testing is based on requirements and specifications” [3]. Black-box testing also called functional testing examines that the software performs all its functions according to the requirements of the customer. The test engineer tests the software by giving the input through the user-interface and checking the expected output. The internal structure of the software is unknown to the tester. The test engineer must be familiar with all the requirements/specifications before test. The following testing types mentioned in [2] come in the category of Black-box testing: boundary value analysis, cause-effect Graphing, random testing, error guessing, regression testing, stress testing, replication testing, data integrating testing, backup and recoverability, configuration testing, performance testing, functional testing, security testing, equivalence partitioning, operational readiness testing, user acceptance testing, compatibility/conversion testing, benchmark testing, usability testing, alpha/beta testing.
1.1.2. White-Box Testing

White-box Testing is "a strategy in which testing is based on the internal paths, structure, and implementation of the software under test (SUT)" [3]. White-box testing also called structural testing [3] examines all the structural paths of the software modules to verify that all the paths including statements, logical decisions have been executed properly. If no statement or logical decision exercised, the test engineer tries to find the reason and possibly write more test cases to exercise un-executed statements. The following testing types mentioned in [2] come in the category of white-box testing: fault insertion, string / module test, error handling, statement coverage, decision coverage, condition coverage, path coverage, data flow coverage, memory leakage and cyclomatic complexity.

There are four levels of testing i.e. unit, integration, system and acceptance testing [3]. Software testing starts at the unit level, continue through integration, system level and ends at acceptance level. There are areas of testing that are hard to test with manual testing i.e. memory leakage, load and performance testing.

Much advancement in the development practices and methodologies have been made in recent years. One of the practices called Component based Software Engineering (CBSE) aimed at reducing the development time is reuse of software capabilities in other software. On the other hand, one way to improve the quality of the software product is to consider the quality aspects that affect the quality of the software i.e. reliability, usability, security and testability. If kept in mind while developing the software, many faults can be eliminated early in the development that can reduce much time spent later on while finding and fixing them. Studies show that "finding and fixing a software problem after delivery is often 100 times more expensive than finding and fixing it during the requirements and design phase"[4].

After developing software, it needs to be tested to verify its intended purpose for which it has been developed. Testing can be performed either manually or automated. Manual testing was done in the past to verify that the system fulfilled the requirements of the customer. Manual testing is a boring activity because the same tasks are performed again and again and it involves human throughout the process so it can be error-prone. While at the same time, it takes much time to write test cases and run them one by one. If a change has to be made in any part of the software then the test cases need to be modified and re-run from the start. So much time has to be spent on testing which increases the cost of the software and most of the times, the software projects are failed to meet the deadlines. Boris Beizer reported that "half the labor expended to develop a working program is typically spent on testing activities" [5]. Today most of the companies are using agile methods i.e. SCRUM, XP as their development methodology and software are being developed in increments. Each increment has to be integrated and
tested with the previous one that increases testing effort of the testers. The test engineer has to write more test cases, run them repeatedly until the final release of the product is delivered. If all testing is performed manually, it will increase testing time which ultimately increases the cost of software testing. One survey about software testing shows that “about 30% to 50% time of a project is spent on testing while another shows about 50% to 75%” [2].

After reading [2, 8], it is assumed that Automated Software Testing (AST) is a process of automating the testing activities which includes development and execution of test scripts to verify test requirements using automated tools for the purposes of improving the STL efficiency. Automating all the tests without any rational is not a good approach because all tests don't have equal importance. So it is necessary to determine some criteria for the possible automatable tests. The best test candidates are those that require more input data, run very often, hard or impossible to run manually and need much effort. The tests, requiring changes in source code frequently, are not beneficial for automation. It is worth to mention that automated testing doesn't aim at finding new defects but to make it sure that the changes made recently in the code didn't introduce new defects. Berner [7] found that the “majority of the new defects are detected by manual tasks not the automated ones because 60% of the bugs are found during an automated testing effort and 80% are found during the development of the tests”. Since automation usually run same tests every time so the chance of skipping any test is zero. Still manual testing can't ignore at all. There are still some areas where we have to perform manual testing. It was reported in an article, published by Computer Science Department of Colorado State University about “Automated Test Software”, “much of testing (software) is still being done manually and the process is intuitively guided” [2]. The demand for automated testing is constantly increasing and companies are moving towards automating their testing processes but no systematic evidence is available which can tell what types of automated testing are reported?

1.2. AUTOMATED TESTING PROCESS
AST refers to the automation of all testing activities of Software Testing Lifecycle (STL). After reading the literature [1, 2, 8], it is assumed that following activities of STL can be automated:

- Test environment setup
- Test development / creation
  - Test data generation
  - Test case generation
- Test execution and results evaluation
1.2.1. Test Environment Setup

Test environment setup includes the activities i.e. configuration of the software, hardware and network. The software includes automated testing tool, framework etc. There are many tools and frameworks available in the market which can be purchased according to the requirements. The hardware is the system on which software should be configured.

1.2.2. Test Development / Creation

Test development consists of test data generation and test case generation.

❖ Test data generation

According to [17], Test data generation is defined as “for a given program element, find a program input on which this element is executed”. Software tester tries to test maximum combination of test data but it doesn’t mean that software is fully tested. There are testing tools available in the market that can show what parts of the program still are not executed? Test data generation helps us to create maximum combination of test data. Different approaches for test data generation have been developed. Few of them are explained below:

- Boundary value analysis
  Boundary value analysis technique partitioned the program domain into input classes. Test data are selected that lies inside input classes and on the boundaries of input classes. Output classes are identified using test data input that causes output at the same manner [20].

- Random approach
  In random test data generation, inputs for the test program are produced randomly until a useful input is found [14].

- Path-oriented approach
  Path-oriented test data generation technique selects paths using control flow information and then generates test data for the selected path(s). It can be dynamic or static in its execution [1]. It has one sub-approach.

  o Constrained-based data generation
    Constrained-based data generation technique is a type of path-oriented data generation approach. It is based on mutation analysis and creates test data using algebraic constraints to find particular types of faults [18]. Mutation analysis is “a fault-based testing method that measures the adequacy of a set of externally created test cases” [16].
• **Goal-oriented approach**
In goal-oriented test-data generation, program input is achieved by executing sequence of sub-goals. This differs from the path-oriented approach in a sense that path stage is eliminated [19]. It has two sub-approaches.

  o **Chaining approach**
The chaining approach for test data generation uses data dependence analysis to identify statements that affect execution of selected statement. These statements then help to generate test data [17].

  o **Assertion-oriented approach**
Assertion-oriented test data generation technique “identifies test cases on which an assertion is violated”. This approach then uncovers errors on test cases at which assertion have been violated [1].

**Test Case Generation**
Test case generation is an activity in which test cases are created. “A test case has an identity and associated with program behavior. It has a number of inputs and a number of expected outputs” [3]. Torkar in [1] make a distinction between test data and test case generation. There are two approaches for test case generation mentioned in [1].

  • **Specification based test case generation and selection**
Specification based test case generation and selection is based on the specification / requirements of the software. Usually specifications are in the form of natural language [1]. One of the advantages of generating tests from specification is to find problems in specifications earlier in the development resulting in saving time and resources [13].

  • **Statistical based test case generation and selection**
Statistical based test case generation and selection is based on statistical computations to identify test cases. It is a combination of several techniques i.e. mutation analysis and genetic algorithms [1].

**1.2.3. Test Execution and Results Evaluation**
Test execution and results evaluation takes place after the tests have created. Test scripts created during test development exercises all tests for unit, integration, system and acceptance testing. The results are recorded for each test. These results are then evaluated against the actual and expected output which clearly shows whether test is passed or failed. It also shows
what decision conditions statements are not exercised? What paths are not executed? What percent code coverage is performed? Etc.

Fig. 1: Automated Testing Process

The objective of investment in test automation is to get financial benefits but if this investment is made without proper consideration/estimation then there is no guarantee that it will give financial gains. Often test automation caused the projects get late due to not enough testing because a large investment of time is required. Companies want to invest in test automation but at the same time they also want some estimation to know how much resources are required and what benefits they will get back in return? Whether will it be beneficial to invest or not? To answer these questions, a need for proper estimation of cost and benefits is
required for test automation. For this proper, a model called Return on Investment (ROI) is used. This ROI model calculates the investment and benefits in term of money. The ROI is usually computed as “the benefits derived divided by the investments made for a given thing” [9]. If test automation is implemented after a thorough analysis, it has many benefits including (tangible and intangible) reduction in time and cost of software testing, improve software quality and faster time to market etc. “A successful automation strategy requires continuous, realistic assessment of the return on investment” [10].

Another model for calculating the ROI is provided in [11, 12]. It explains that if the cost of automated testing is less than the total cost of performing the same tests manually then the investment is giving a payback. This model assumes that the benefits for automated and manual testing are equal. It doesn’t discuss any intangible benefits of automated testing i.e. improved quality, early time to market, better use of resources, increase confidence etc. Ramler R, Wolfmaier K. criticized this model in [6] and explained some reasons why this model can’t give accurate analysis for estimating the investment in test automation. Dustin in [2] encourages that “if an organization wants to introduce AST to their project, they should develop a business case in which they identify the business need and justify it in terms of cost and benefits”. One of the main factors that affect the cost of testing is the effort spent on executing the test cases. In the case of AST, effort increases in the beginning due to setting up the automated environment, writing test cases and test scripts etc. But later on, minimum effort is required for the execution of these tests. The system can execute all the tests with no interruption until the execution finished. Manual testing requires much effort particularly to execute test cases. This effort then translates into cost of testing. What is the effort difference between automated and manual testing is questionable. It is worth to mention here that automated testing may not provide financial gains at the start because cost of automation tool, staff training and configuration of the test environment is added. But as the number of run increases, the financial benefits start to be visible.

1.3. PURPOSE
The purpose of this thesis is to conduct a systematic review and to find & summarize all the relevant data related to research questions. “A systematic literature review is a means of identifying, evaluating and interpreting all available research relevant to a particular research question or topic area or phenomenon of interest. Individual studies contributing to a systematic review are called primary studies; a systematic review is a form a secondary study” [15].
1.4. RESEARCH QUESTIONS
The research questions, which will be answered in this systematic review, are presented below:
   1. What types of automated testing are reported?
   2. How many industrial case studies can be found for each type of automated testing?
   3. Can we measure the ROI for automated testing?
   4. Can we measure the difference between manual and automated testing in terms of effort estimation?

1.5. THESIS STRUCTURE
This section provides an overview of thesis structure as follows:
   - Background
   This section provides detailed background information about automated testing, automated testing process, effort estimation and ROI for automated testing to understand the need for conducting the systematic review. This section also provides research questions that will be answered in this review.
   - Systematic Review Method
   This section provides detailed description of the systematic review. A review protocol will be presented which provides elements in detail to conduct the review.
   - Results and Analysis
   This section presents the results and detailed analysis of the data found related to the research questions. Some statistical analysis of data and detail descriptive results of this review will be presented.
   - Discussions
   This section presents discussion on the basis of results and gives my point of view for different results.
   - Summary of findings
   This section provides detailed summary of the findings of this systematic review as well as findings of un-structured search.
   - Conclusions & Future work
   Finally, this section presents conclusions of this study and includes some future work.
2. SYSTEMATIC REVIEW METHOD

2.1. SEARCH STRATEGY
The search strategy is based on the exploration of electronic databases of journal articles and conference proceedings. For this purpose, the author has included “Inspec” electronic database in the search strategy. This database is selected because it also searches data from other databases i.e. IEEE Xplore, ACM digital Library etc and is more relevant to find research work related to research questions. “A general approach is to break down the question into individual facets i.e. population, intervention, outcomes, study designs. Then draw up a list of synonyms, abbreviations, and alternative spellings” [15]. Taking this into account, the author has broken down the questions into individual facets and extracted some keywords/terms from the research questions to search the primary study. After performing the preliminary search on the selected keywords, there has been found research articles related to questions and the author extracted more keywords i.e. synonyms. The keywords and synonyms were then used together in the search strings with Boolean operators “AND” and “OR”. The following keywords are chosen for the search query. Keywords have been divided into four distinct numbers for ease of understanding.

1. software, application, program, system
2. automated, automation, automatic, automating
3. testing, test
4. cost, time, effort estimation, effort, ROI, benefit*, return on investment, cost-benefit analysis, defect, error, fault, bug, multithread*, concurrent

2.2. STUDY SELECTION
Study selection is a process of identifying “those primary studies that provide direct evidence about the research question” [15]. The study selection process is a two stage procedure which will define the selection criterion to include the more relevant studies to our literature review. First, the author study the document in two perspectives i.e. document title and abstract / introduction / keywords to see relevance of the document related to research questions. If it provides evidence of AST then that study will be selected for further investigation. Second, more refined form of selection will be used where only those studies will be included in the review which fulfills the following inclusion or exclusion criteria:
2.2.1. **Inclusion Criteria**

The research papers / articles or conference proceedings paper which should:

1. Provide different automated testing types i.e.
   
   I. Code-based testing
   II. Specification-based testing
   III. Model-based Testing
   IV. Security testing
   V. Fault-based testing
   VI. Functional testing
   VII. Unit testing
   VIII. Module testing
   IX. Integration Testing
   X. System testing
   XI. Acceptance testing
   XII. Regression testing
   XIII. GUI testing
   XIV. Code Coverage testing
   XV. Random testing
   XVI. Performance testing
   XVII. Stress testing
   XVIII. Load testing
   XIX. Memory leakage testing
   XX. Usability Testing
2. Provide AST benefits
3. Provide ROI calculation, cost benefit analysis for test automation, cost estimation, effort estimation for automated or manual testing.
4. Be in English language.

2.2.2. **Exclusion Criteria**

The research papers/ conference proceedings which should:

1. Have keywords stated-above but don’t provide evidence about automated testing.
2. Be in other language.

2.3 **STUDY QUALITY ASSESSMENT**

After the selection of the primary studies, there is a need to assess the quality of each primary study. There is much literature available which guides how to assess the quality of the studies. The CRD Guideline [15] suggests “using an assessment of study design to guarantee a minimum
level of quality”. There are two types of research studies called qualitative and quantitative. The author doesn’t restrict any type of study to include in the quality assessment. We perform this quality assessment of study to provide more detailed inclusion and exclusion criteria and as guidance for data synthesis and interpretation of findings and results [15].

Detailed quality assessment of primary studies is “usually based on quality instruments which are checklists of factors that need to be assessed for each study” [15]. The author has created a checklist of items to assess the quality of the data. If all the items in the checklist provide the answer with (Yes), then the study will be added to the review process. Below the checklist of quality items:

<table>
<thead>
<tr>
<th>Sr. #</th>
<th>Quality items checklist</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Does the document have proper introduction about the topic including aims and objectives?</td>
</tr>
<tr>
<td>2</td>
<td>Does the paper explicitly account for the research contributions presented in the paper?</td>
</tr>
<tr>
<td>3</td>
<td>Are the results or conclusions of the research discussed?</td>
</tr>
<tr>
<td>4</td>
<td>Do the conclusions fulfill the aims and objectives of the research?</td>
</tr>
<tr>
<td>5</td>
<td>Are the references for data sources are given properly?</td>
</tr>
</tbody>
</table>

Table 2.1 Quality Item Checklist

2.4. DATA EXTRACTION

Data extraction is a way of extracting the relevant information from the primary studies needed to answer the research questions. For this purpose, the author has designed a data extraction form to collect that information. The data extraction form has the following information:

<table>
<thead>
<tr>
<th>Article information</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. Name of article</td>
</tr>
<tr>
<td>b. Author name(s)</td>
</tr>
<tr>
<td>c. Name of the journal or conference proceedings, symposium and workshop in which the article is published or presented.</td>
</tr>
<tr>
<td>d. Date of publication</td>
</tr>
<tr>
<td>e. Page numbers if available</td>
</tr>
<tr>
<td>f. Volume number if available</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methodology used for research</th>
<th>Study Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. Experiment</td>
<td>a. Academic</td>
</tr>
<tr>
<td>b. Experience</td>
<td>b. Industrial</td>
</tr>
<tr>
<td>c. Case study</td>
<td></td>
</tr>
</tbody>
</table>
d. Survey

**Article’s area of study relevant to research questions**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a.</td>
<td>Automated testing types</td>
</tr>
<tr>
<td>b.</td>
<td>Number of industrial case studies for each testing type</td>
</tr>
<tr>
<td>c.</td>
<td>Methods for return on investment (ROI) or cost-benefit analysis for test automation</td>
</tr>
<tr>
<td>d.</td>
<td>Benefits of automated testing</td>
</tr>
<tr>
<td>e.</td>
<td>Effort Estimation</td>
</tr>
<tr>
<td>i. Manual testing effort estimation</td>
<td></td>
</tr>
<tr>
<td>ii. Automated testing effort estimation</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2: Extraction of Data Items

If the author finds duplicate copies of the primary studies, then he will use the copies with the latest date if it provides complete information and full text i.e. introduction, study methodology, results and conclusion.

### 2.5. DATA SYNTHESIS

Data synthesis activity includes "collecting and summarizing the results of the selected primary studies" [15]. The aim of this review is to collect the data about research questions and summarize results in a systematic way. After extraction of the data by using data extraction form, it will be analyzed to answer the questions. The results will be summarized and presented in the form of tables and graphs. The following tables will be used to summarize the data.

Table 2.3: Data Synthesis form

<table>
<thead>
<tr>
<th>Sr. #</th>
<th>Automated Testing Types</th>
<th>Number of studies</th>
<th>Article Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 2.4: Data Synthesis form

<table>
<thead>
<tr>
<th>Sr. #</th>
<th>Benefits</th>
<th>Article #</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3. RESULTS AND ANALYSIS
In this section, the data will be extracted related to research questions, present descriptive results and their statistical analysis. Before results, the procedure of study selection process is presented.

3.1. SELECTION OF STUDIES
This section gives the detailed description of the procedure for the selection of the relevant primary studies related to research questions. We have decided to search the primary studies from the year 1998-2013 to see how many studies were conducted related to our search questions. We formed the search query into two steps to select the most relevant studies for our research questions.

<table>
<thead>
<tr>
<th>Steps</th>
<th>Formation of query</th>
<th>Number of articles</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(&quot;software&quot; OR &quot;application&quot; OR &quot;program&quot; OR &quot;system&quot;) AND (&quot;automated&quot; OR &quot;automation&quot; OR &quot;automatic&quot; OR &quot;automating&quot;) AND (&quot;testing&quot; OR &quot;test&quot;) AND (&quot;cost&quot; OR &quot;time&quot; OR &quot;effort estimation&quot; OR &quot;effort&quot; OR &quot;ROI&quot; OR &quot;benefit*&quot; OR &quot;return on investment&quot; OR &quot;cost-benefit analysis&quot; OR &quot;defect&quot; OR &quot;error&quot; OR &quot;fault&quot; OR &quot;bug&quot; OR &quot;concurrent&quot; OR &quot;multithread*&quot;))</td>
<td>17574</td>
</tr>
<tr>
<td>2</td>
<td>(((&quot;software&quot; OR &quot;application&quot; OR &quot;program&quot; OR &quot;system&quot;) AND (&quot;automated&quot; OR &quot;automation&quot; OR &quot;automatic&quot; OR &quot;automating&quot;) AND (&quot;testing&quot; OR &quot;test&quot;) AND (&quot;cost&quot; OR &quot;time&quot; OR &quot;effort estimation&quot; OR &quot;effort&quot; OR &quot;ROI&quot; OR &quot;benefit*&quot; OR &quot;return on investment&quot; OR &quot;cost-benefit analysis&quot; OR &quot;defect&quot; OR &quot;error&quot; OR &quot;fault&quot; OR &quot;bug&quot; OR &quot;concurrent&quot; OR &quot;multithread&quot;) )) AND [English] WN LA AND ([program testing] OR [automatic testing] OR [program verification] OR [formal verification] OR [internet] OR [software tools]) WN CV)</td>
<td>3270</td>
</tr>
</tbody>
</table>

Table 3.1: selection of articles

The first step gave total 17574 articles which was hard to read and extract data. In addition, it included articles that were not mostly relevant to research questions. In step two, author used the “Inspec” database “refine result” facility named “controlled vocabulary” to minimize the search by adding most relevant “keywords” into the query i.e. articles that are in English and have keywords program testing, automated testing, program verification, formal verification, internet, software tools in controlled vocabulary. This minimized the searched articles to 3270 and excluded 14304 articles from the first step. After thorough analysis of the articles based on title, abstract and conclusion, 58 articles were selected from 3270 that were more relevant and fulfilled most of quality assessment criteria. The author couldn’t find many articles about effort.
estimation through the query. Therefore, a manual search has been performed. This search consists of exploring the articles of conference proceedings titled “Software Testing Verification and Validation” during 1998-2013 to find more article about effort estimation. The selection of this conference proceeding is based on the fact that most of the articles, found through query, have this conference proceeding sources. Two more articles have been found after searching above mentioned conference proceeding. The following table shows the selected articles with their title and year.

<table>
<thead>
<tr>
<th>Sr. #</th>
<th>Article Name</th>
<th>Year</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A strategy for using genetic algorithms to automate branch and fault-based testing</td>
<td>1998</td>
<td>[23]</td>
</tr>
<tr>
<td>2</td>
<td>Applying test automation to type acceptance testing of telecom networks: a case study with customer participation</td>
<td>1999</td>
<td>[54]</td>
</tr>
<tr>
<td>3</td>
<td>UML-based integration testing</td>
<td>2000</td>
<td>[60]</td>
</tr>
<tr>
<td>4</td>
<td>Module testing embedded software-an industrial pilot project</td>
<td>2001</td>
<td>[32]</td>
</tr>
<tr>
<td>5</td>
<td>Automating software module testing for FAA certification</td>
<td>2001</td>
<td>[31]</td>
</tr>
<tr>
<td>6</td>
<td>Automated test case generation for the stress testing of multimedia systems</td>
<td>2002</td>
<td>[75]</td>
</tr>
<tr>
<td>7</td>
<td>A modular tool for automated coverage in software testing</td>
<td>2003</td>
<td>[21]</td>
</tr>
<tr>
<td>8</td>
<td>Framework and model for automated interoperability test and its application to ROHC</td>
<td>2003</td>
<td>[71]</td>
</tr>
<tr>
<td>9</td>
<td>&quot;DART: a framework for regression testing &quot;&quot;nightly/daily builds&quot;&quot; of GUI applications&quot;</td>
<td>2003</td>
<td>[43]</td>
</tr>
<tr>
<td>10</td>
<td>JMLAutoTest: a novel automated testing framework based on JML and JUnit</td>
<td>2003</td>
<td>[65]</td>
</tr>
<tr>
<td>11</td>
<td>DART: distributed automated regression testing for large-scale network applications</td>
<td>2004</td>
<td>[44]</td>
</tr>
<tr>
<td>12</td>
<td>TestEra: specification-based testing of Java programs using SAT</td>
<td>2004</td>
<td>[64]</td>
</tr>
<tr>
<td>13</td>
<td>A model-based approach to the security testing of network protocol implementations</td>
<td>2006</td>
<td>[55]</td>
</tr>
<tr>
<td>14</td>
<td>Coverage-based testing on embedded systems</td>
<td>2007</td>
<td>[28]</td>
</tr>
<tr>
<td>15</td>
<td>Automated testing of timeliness : a case study</td>
<td>2007</td>
<td>[59]</td>
</tr>
<tr>
<td>16</td>
<td>Model-based security vulnerability testing</td>
<td>2007</td>
<td>[56]</td>
</tr>
<tr>
<td>17</td>
<td>RPB in software testing</td>
<td>2007</td>
<td>[74]</td>
</tr>
<tr>
<td>18</td>
<td>JAT: a test automation framework for multi-agent systems</td>
<td>2007</td>
<td>[50]</td>
</tr>
<tr>
<td>19</td>
<td>Test effort estimation models based on test specifications</td>
<td>2007</td>
<td>[80]</td>
</tr>
<tr>
<td>20</td>
<td>Combined symbolic and concrete execution of TTCN-3 for automated testing</td>
<td>2008</td>
<td>[29]</td>
</tr>
<tr>
<td>22</td>
<td>A test automation solution on GUI functional test</td>
<td>2008</td>
<td>[39]</td>
</tr>
<tr>
<td></td>
<td>Title</td>
<td>Year</td>
<td>Pages</td>
</tr>
<tr>
<td>---</td>
<td>----------------------------------------------------------------------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>23</td>
<td>Automated usability testing using HUI analyzer</td>
<td>2008</td>
<td>[47]</td>
</tr>
<tr>
<td>24</td>
<td>Automated acceptance testing using fit</td>
<td>2009</td>
<td>[53]</td>
</tr>
<tr>
<td>26</td>
<td>Hermes: a tool for testing mobile device applications</td>
<td>2009</td>
<td>[33]</td>
</tr>
<tr>
<td>27</td>
<td>On-the-fly model-based testing of Web services with Jambition</td>
<td>2009</td>
<td>[61]</td>
</tr>
<tr>
<td>28</td>
<td>Automated GUI Testing for J2ME Software Based on FSM</td>
<td>2009</td>
<td>[58]</td>
</tr>
<tr>
<td>29</td>
<td>Idea: automatic security testing for Web applications</td>
<td>2009</td>
<td>[37]</td>
</tr>
<tr>
<td>30</td>
<td>On the effectiveness of unit test automation at Microsoft</td>
<td>2009</td>
<td>[49]</td>
</tr>
<tr>
<td>31</td>
<td>An Alternative Approach to Test Effort Estimation Based on Use Cases</td>
<td>2009</td>
<td>[77]</td>
</tr>
<tr>
<td>32</td>
<td>A Simple Approach for Estimation of Execution Effort of Functional Test cases</td>
<td>2009</td>
<td>[78]</td>
</tr>
<tr>
<td>33</td>
<td>Estimating manual test execution effort and capacity based on execution points</td>
<td>2009</td>
<td>[79]</td>
</tr>
<tr>
<td>34</td>
<td>Automated Software Testing using Metaheuristic Technique based on an Ant Colony Optimization</td>
<td>2010</td>
<td>[22]</td>
</tr>
<tr>
<td>35</td>
<td>Using meta-data technique for component based black box testing</td>
<td>2010</td>
<td>[34]</td>
</tr>
<tr>
<td>36</td>
<td>Automated GUI Test Coverage Analysis Using GA</td>
<td>2010</td>
<td>[41]</td>
</tr>
<tr>
<td>37</td>
<td>Scenario-Based Approach for Blackbox Load Testing of Online Game Servers</td>
<td>2010</td>
<td>[72]</td>
</tr>
<tr>
<td>38</td>
<td>Integrating model-based testing with evolutionary functional testing</td>
<td>2010</td>
<td>[64]</td>
</tr>
<tr>
<td>39</td>
<td>A Hybrid Approach for Model-based Random Testing</td>
<td>2010</td>
<td>[63]</td>
</tr>
<tr>
<td>40</td>
<td>A reactivity-based framework of automated performance testing for web applications</td>
<td>2010</td>
<td>[73]</td>
</tr>
<tr>
<td>41</td>
<td>Automating Java program testing using OCL and AspectJ</td>
<td>2010</td>
<td>[46]</td>
</tr>
<tr>
<td>42</td>
<td>Automated Behavioral Regression Testing</td>
<td>2010</td>
<td>[42]</td>
</tr>
<tr>
<td>43</td>
<td>Coverage Criteria for Automatic Security Testing of Web Applications</td>
<td>2010</td>
<td>[38]</td>
</tr>
<tr>
<td>44</td>
<td>Automated web application testing using search based software engineering</td>
<td>2011</td>
<td>[27]</td>
</tr>
<tr>
<td>45</td>
<td>Automatic test data generation for path testing using genetic algorithms</td>
<td>2011</td>
<td>[26]</td>
</tr>
<tr>
<td>46</td>
<td>Invariant-Based Automatic Testing of Modern Web Applications</td>
<td>2011</td>
<td>[40]</td>
</tr>
<tr>
<td>47</td>
<td>Software automated testing: A solution to maximize the test plan coverage and to increase software reliability and quality in use</td>
<td>2011</td>
<td>[76]</td>
</tr>
<tr>
<td>48</td>
<td>Experiences of System-Level Model-Based GUI Testing of an Android Application</td>
<td>2011</td>
<td>[57]</td>
</tr>
</tbody>
</table>
49 Automated test data generation for mutation testing using AspectJ programs 2011 [69]

50 DART: Directed Automated Random Testing 2011 [47]


52 Using unfoldings in automated testing of multithreaded programs 2012 [68]

53 A novel approach of automation testing on mobile devices 2012 [36]


55 JSART: JavaScript assertion-based regression testing 2012 [45]

56 CAST: automating software tests for embedded systems 2012 [66]

57 AutoBlackTest: Automatic Black-box Testing of Interactive Applications 2012 [52]

58 Automated system testing using visual GUI testing tools: A comparative study in industry 2012 [51]

59 Extension of Selenium RC tool to perform automated testing with databases in web applications 2013 [35]

60 Automatic test generation for mutation testing on database applications 2013 [70]

Table 3.2 Selected articles with titles and years

3.2. ANALYSIS OF SELECTED ARTICLES

Number of selected articles over the period 1998-2013

Graph 3.1 shows the number of selected articles / studies during the period 1998-2013. This graph clearly shows that most of the articles are during the period 2007-2012 i.e. 47 articles while less numbers are during 1998-2006 i.e. 13 articles. The highest number of studies was carried out in 2009 and 2010 i.e. 10 studies per year while no study was carried out in 2005 i.e. zero studies. The average studies conducted during 2007-2013 is approximately 6.71 and during 1998-2006 is approximately 1.44 per year. The years 1998, 1999, 2000, 2002 and 2006 have only one study recorded while year 2001, 2004 and 2013 have two studies per year.
Research methodology used in selected Articles

Graph 3.2 shows the research methodology in which studies were conducted. The highest number of studies is experimental studies. Case studies come at the second place while experience and survey studies are at place 3 and 4 respectively.
Number of Articles per Study Environment

Graph 3.3 shows the number of selected articles relative to their study environment. Graph shows that most the studies were conducted in academic environment while less studies in Industry. It also gives information that the Academic Institutions are big source of research. Graph 3.3 shows that industrial studies dominated in the start i.e. during 1999-2001 while not much work was done in industry during 2002-2008. Most of the industrial studies were conducted during 2009-2012 i.e. 10 articles. Academic studies during period 2007-2012 don't have much difference in numbers expect year 2009.
Graph 3.3: Number of Selected Articles per Study Environment

Graph 3.4: Number of Selected Articles per Study Environment over the period 1998-2013
3.3. AUTOMATED TESTING TYPES
After thorough analysis of the research papers, the author found following automated testing types that were reported in the literature.

<table>
<thead>
<tr>
<th>Sr. #</th>
<th>Automated Testing Types</th>
<th>Number of studies</th>
<th>Article Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Code-Coverage Testing</td>
<td>10</td>
<td>[21,22,23,24,25,26,27,28,29,30]</td>
</tr>
<tr>
<td>2</td>
<td>Model-based Testing</td>
<td>10</td>
<td>[55,56,57,58,59,60,61,62,62,64]</td>
</tr>
<tr>
<td>3</td>
<td>Functional Testing</td>
<td>7</td>
<td>[33,34,35,36,37,61,64]</td>
</tr>
<tr>
<td>4</td>
<td>GUI Testing</td>
<td>6</td>
<td>[33,39,40,41,57,58]</td>
</tr>
<tr>
<td>5</td>
<td>Fault-based Testing</td>
<td>5</td>
<td>[23,46,59,69,70]</td>
</tr>
<tr>
<td>6</td>
<td>Security Testing</td>
<td>4</td>
<td>[37,38,55,56]</td>
</tr>
<tr>
<td>7</td>
<td>Regression Testing</td>
<td>4</td>
<td>[42,43,44,45]</td>
</tr>
<tr>
<td>8</td>
<td>Random Testing</td>
<td>4</td>
<td>[46,47,62,63]</td>
</tr>
<tr>
<td>9</td>
<td>Integration Testing</td>
<td>3</td>
<td>[50,60,66]</td>
</tr>
<tr>
<td>10</td>
<td>Specification-based Testing</td>
<td>3</td>
<td>[65,66,67]</td>
</tr>
<tr>
<td>11</td>
<td>System Testing</td>
<td>3</td>
<td>[50,51,52]</td>
</tr>
<tr>
<td>12</td>
<td>Module Testing</td>
<td>2</td>
<td>[31,32]</td>
</tr>
<tr>
<td>13</td>
<td>Stress Testing</td>
<td>2</td>
<td>[74,75]</td>
</tr>
<tr>
<td>14</td>
<td>Unit Testing</td>
<td>2</td>
<td>[49,50]</td>
</tr>
<tr>
<td>15</td>
<td>Acceptance Testing</td>
<td>2</td>
<td>[53,54]</td>
</tr>
<tr>
<td>16</td>
<td>Concurrency Testing</td>
<td>1</td>
<td>[68]</td>
</tr>
<tr>
<td>17</td>
<td>Interoperability Testing</td>
<td>1</td>
<td>[71]</td>
</tr>
<tr>
<td>18</td>
<td>Load Testing</td>
<td>1</td>
<td>[72]</td>
</tr>
<tr>
<td>19</td>
<td>Usability Testing</td>
<td>1</td>
<td>[48]</td>
</tr>
<tr>
<td>20</td>
<td>Performance Testing</td>
<td>1</td>
<td>[73]</td>
</tr>
<tr>
<td>21</td>
<td>Memory Leakage Testing</td>
<td>1</td>
<td>[76]</td>
</tr>
<tr>
<td>22</td>
<td>Environment Testing</td>
<td>1</td>
<td>[77]</td>
</tr>
</tbody>
</table>

Table 3.3: Number of articles for each testing type

Number of articles for each testing type
Graph 3.5 shows the number of articles for each testing type. The highest number of articles is about Code Coverage Testing and Model-based Testing i.e. 10 articles. This also shows that most of the research was carried out in code coverage testing and model-based testing. The less number of articles is about Usability Testing, Concurrency Testing, Interoperability Testing, Load Testing, Performance Testing, Memory Leakage Testing, Environment Testing i.e. 1 article each. This shows that researchers are not much interested in these testing types. Functional Testing, GUI Testing and Fault-based Testing come at the second, third and fourth place i.e. 7, 6, 5 articles respectively. We found 4 articles for each Security Testing, regression testing and random Testing while 3 articles for each Integration Testing, System Testing and Specification-

Graph 3.5 Number of articles for each testing type

- Environment Testing: 1 article
- Memory Leakage Testing: 1 article
- Stress Testing: 2 articles
- Performance Testing: 1 article
- Load Testing: 1 article
- Interoperability Testing: 1 article
- Fault-based Testing: 5 articles
- Concurrency Testing: 3 articles
- Specification-based Testing: 3 articles
- Modal-based Testing: 10 articles
- Acceptance Testing: 2 articles
- System Testing: 3 articles
- Integration Testing: 3 articles
- Unit Testing: 2 articles
- Usability Testing: 1 article
- Random Testing: 4 articles
- Regression Testing: 4 articles
- GUI Testing: 6 articles
- Security Testing: 4 articles
- Functional Testing: 7 articles
- Module Testing: 10 articles
- Code Coverage Testing: 10 articles

(number of articles)
**Number of articles over the period 1998-2013**

Graph 3.6 shows each testing type over the period 1998-2013. It is worth to mention here that we found most of the articles during the period 2007-2012. Graph shows that Code coverage testing and model-based Testing were most popular during the period 2007-2012 while functional testing and GUI testing during 2008-2012. We found some articles about code coverage testing in the year 1998, 2003 and model based testing in the year 2000. We can say that research in code coverage testing and model-based testing is not new. We found only 2 articles about Module testing in the year 2001. It shows that module testing was not popular among researchers. Fault-based Testing have been reported in 5 articles. It started in 1998 and we found a gap during 1999-2006. It again started in 2007 and researchers are still interested to investigate it. Regression Testing and Specification-based Testing started to be investigated by researchers in 2003 and then have a gap during 2005-2009 for regression testing and 2005-2011 for specification-based Testing. The overall analysis shows that automated testing started to get popularity from 2006. It also shows that most of the testing types were popular during 2006-2012 i.e. Code coverage testing, model-based testing, fault-based testing, functional testing, security testing, GUI testing, regression testing, random Testing, Unit Testing, Integration Testing and System Testing.
Graph 3.6: Number of articles over the period 1998-2013 for each Testing Type
- **Code Coverage testing**

The purpose of code coverage testing is to "measures the percentage of a system's code that is being exercised by a test suite" [2]. Coverage based testing is important because it shows what parts of the code are executed and what parts are remaining in terms of statement coverage, branch coverage and path coverage.

The author found 10 research articles reporting automated coverage based testing. Eugenia Díaz et al. [21] have presented a tool which can implement three algorithms to obtain branch coverage i.e. Tabu Search algorithm, scatter search algorithm and random algorithm for programs written in C++. This tool automatically generates test cases using any of the above algorithms. Praveen Ranjan Srivastava and Km Baby [22] have proposed a technique called ACO (Ant Colony Optimization) for the automated full coverage of state transitions in a system. The authors implemented this technique in an algorithm to generate optimal test sequences based on State transition diagrams. These sequences are then used to automatically generate test cases that cover all the transitions in a program at least once.

B. F. Jones et al. [23] have used genetic algorithms for automatic branch coverage in software testing. The authors used genetic algorithms to derive test sets that execute every branch of the program. Debasis Mohapatra et al. [24] used genetic algorithms for the optimization of test cases for path testing. According to the authors, they haven’t found optimized test suits using genetic algorithms. For this purpose, they used the sampling technique to select the optimized test suits. Gentiana Ioana Latiu et al. [25] have compared three evolutionary algorithms named Particle Swarm Optimization, Simulated Annealing and genetic algorithms to automatically generate tests for path testing. Wang Xibo and Su Na [26] have discussed about genetic algorithms and designed a system that automatically generates test data for path testing using genetic algorithms.

Nadia Alshahwan and Mark Harman [27] have described an automated approach for branch coverage using search based software engineering. The authors used an algorithm which uses Korel's Alternating Variable Method (AVM), constant seeding and dynamically mined values algorithms. A tool was developed to implement this algorithm. X. Wu et al. [28] have described an automatic code-coverage testing for embedded systems. Most of the articles, described above, focused on code instrumentation technique but authors focused on code instrumentation minimization in this article. A tool was developed which had four main components i.e. automatic code inspector, program instrumentation module, automatic test execution platform and analysis module. Xuezhi Xing et al. [29] have proposed a framework for automatic testing which executes TTCN-3 language test scripts using symbolic and concrete execution to automatically generate test data for path coverage. Ahmed S. Ghiduk and Mary
Jean Harrold [30] have presented a testing technique which uses genetic algorithms to automatically generate test data for data-flow coverage.

- **Module/String Testing**
  A module is a group of functions. Module Testing examines “the related group of modules that constitutes a software function. It determines whether modules work successfully to form a cohesive unit” [8].

The author found 2 articles reporting automated module testing. Usha Santhanam [31] has reported an automated module testing for program written in Ada. The author performed module testing using a combination of tool set called Test Set Editor (TSE). The tool set performs three automated tasks: identify input and output for the module, creates test driver for the module and measures test coverage. All process is automatic expect the test case generation. Jason McDonald et al. [32] have reported automated module testing for embedded distributed real time systems. Four modules were tested in total which had different functions and the criteria for testing each module were discussed in detail i.e. test case selection, test execution and reporting.

- **Functional Testing**
  Functional testing is to verify “that the system meets the functional requirements” [2]. The purpose of functional testing is to “assess whether the application does what it is supposed to do in accordance with specified requirements” [8].

The author found 5 articles reporting automated functional testing. Sakura She et al. [33] have developed an automated testing framework called Hermes to test the J2ME based mobile applications. This framework is based on client-server architecture where client provides the facility to write test cases, send them to server-side for execution and the server executes them and sends the results back to the client. This framework supports to run multi-facet test cases i.e. functional, GUI and environment. Furqan Naseer et al. [34] have presented a technique for the automated functional testing for the testing of components using meta-data technique. In this paper, authors have used the reverse engineering approach to get the meta-data information of a component. A tool has developed to implement this approach. Test data can be entered manually or generated randomly. Andreza M. F. V. de Castro et al. [35] have presented a solution for automatic functional testing of web applications with databases. For this purpose, authors extended one open source automatic functional testing tool called Selenium RC. The extensions were made by adding new assert functions in the selenium framework. Leckraj Nagowah and Gayeree Sowamber [36] have presented a test automation framework for automated testing of mobile phones devices. This framework is based on data-
driven framework and has three layers i.e. abstract layer, reporting layer and data layer. Abstract layer consists of object repository, test data and test executor components. Reporting layer consists of report generator component and data layer consist of test data which is stored in the database. ZHU Xiaochun et al. [39] have presented a solution for automated GUI functional testing (See section GUI Testing).

- **Security Testing**

Security testing is to verify “that the system fulfills the security requirements” [2]. The purpose of security testing “involves checks to verify the proper performance of system access and data access mechanisms” [8].

We found 4 articles reporting automated security testing. Thanh-Binh Dao and Etsuya Shibayama [37] have proposed an automatic testing method for the security vulnerabilities found in web applications. The proposed method first generates automatic attack requests with malicious data from different sources and then performs automatic security checking using dynamic tainting technique to detect this data. It further generates automated test data using constraint-based test data generation technique to execute unexecuted branches of the program. Thanh Binh Dao and Etsuya Shibayama [38] have proposed a technique for automated security testing based on three types of coverage criteria i.e. wrapper coverage, vulnerability-aware wrapper coverage and vulnerability-aware sink coverage. These types of coverage are basically test case creation methods for different types of coverage i.e. branch coverage, statement coverage. William H. Allen et al. [55] and Percy A. Pari Salas et al. [56] have presented a model-based approach for automated security testing (see section Model-based Testing).

- **GUI Testing**

GUI testing is the testing of application’s user interface to verify that the application is functionally correct. The purpose of GUI testing is to check that the GUI objects, functional flow among them and the output is the same as expected [39, 82].

The author found 4 article reporting automation GUI testing. ZHU Xiaochun et al. [39] have presented an automated solution for GUI functional testing. This automated solution has four layers i.e. test case management dashboard, test driver, execution engine and reporting engine. For test case design, keyword-driven approach was defined to specify test steps in a formal way. An important part of this solution is test driver. Test driver interprets test cases into test scripts which are class objects and send them to execution engine for running. Ali Mesbah et al. [40] have proposed a method to automatically test Ajax based web applications using crawling technique to extract state flow graph from web applications. Two algorithms were defined for
this purpose. These algorithms examine the DOM-tree to look for different elements that can cause the change in the state of the web application in case of events. This process continues until all states are covered. The authors used the constraints implemented in the web pages as an oracle to check different states. Two types of invariants were discussed i.e. generic invariants and application specific invariants. Abdul Rauf et al. [41] have proposed a technique for automated GUI testing based on genetic algorithms. In the paper, authors used genetic algorithms for GUI coverage analysis and optimization of test paths. Sakura She et al. [33] have developed an automated testing framework called Hermes to test the J2ME based mobile applications (see section functional Testing).

- **Regression Testing**
  The purpose of regression testing is to verify “that the action performed to fix the software doesn’t, in turn, create a new error in another area of the software system” [8].

The author found 4 articles reporting automated regression testing. Wei Jin et al. [42] have proposed a novel automated approach called BEnavioral Regression Testing (BERT). The approach provided by the authors focuses only the changed parts of the software code. It generates test data for those parts which have been changed / modified recently and apply generated test data on both versions i.e. new and old version, then analysis the differences and report to the developers. Smoke testing / daily builds is the testing of basic functionality of the system. Atif Memon et al. [43] have described a framework called DART (Daily Automated Regression Tester) for the automated regression testing of GUI software. The process of DART consists of 12 steps together with 6 developer / tester steps referred as phases. The 12 steps of DART include Rip AUT’s GUI, create event flow graphs and integration tree, create matrix M which describe the number of test cases of specific length on specific component, generate test cases, generate expected output, instrument code, execute test cases and compare with expected output, generate execution report, generate coverage report, email reports to developers, generate additional test cases, generate additional expected output.

Brent N. Chun [44] has designed and implemented a prototype framework called DART (Distributed Automated Regression Testing) for the automated regression testing of distributed applications. This framework provides users with the facility to write and execute distributed tests of multiple type at large scale. The automated architecture of DART consists of 6 components. These components are 1) network topology 2) remote execution and file transfer, 3) scripting and programming environment, 4) preprocessing, execution and postprocessing, 5) fault injection and 6) performance anomaly injection. Shabnam Mirshokraie and Ali Mesbah [45] proposed a technique called JAVASCRIPet Assertion-based regression testing (JSART) for automated regression testing of JavaScript in web applications. This technique is based on the
idea of performing dynamic analysis on source code and extracting invariants. These invariants are then used as runtime assertions to uncover regression faults after changing the source code subsequently for web application under test. This technique has four steps 1) JavaScript tracing 2) Invariant generation 3) Filtering unstable assertions 4) Regression testing through assertions.

- **Random Testing**
  Random testing is the testing of a program by giving the random input and comparing the expected output with specifications.

The author found 4 articles reporting automated random testing. Yoonsik Cheon and Carmen Avila [46] have proposed an automated testing approach for Java programs by combining random testing and OCL (Object Constraint Language). First test data is generated randomly for each method, and then OCL is used for two purposes 1) To select the test data that fulfills the method’s pre-conditions 2) to compare the output with method’s post-conditions. Patrice Godefroid et al. [47] have presented an approach called Directed Automated Random Testing (DART). The authors used the static source code parsing technique for interface extraction of a program. Using this interface, they generated the test driver and performed random testing. By performing the dynamic analysis of program behavior, more input data is generated automatically for random testing. Muhammad Zohaib Iqbal et al. [62] have proposed a model-based approach for automated random testing of embedded real time systems (see section model-based testing). Stefan Mohacsi and Johannes Wallner [63] have proposed a model-based approach for automated random testing (see section model-based testing).

- **Usability Testing**
  Usability is one of the quality attributes which should be considered highly when developing interactive systems. Usability testing ensures that the system is easy to use and users of a system can carry out the intended tasks efficiently, effectively and satisfactorily [48].

The author found 1 article reporting automated usability testing. Simon Baker et al. [48] have presented an overview of automated usability testing tool called HUI (Handheld User Interface) analyzer. This tool takes three inputs, perform analysis and shows results in the form of report. First it performs comparison checking which compares expected action sequence (EAS) and actual action sequence (AAS) and highlights usability problems. Second, it performs assertion checking on the static properties of GUI objects e.g. font size. Third it performs hotspot analysis which compares the relative use of GUI objects with colors.
• **Unit Testing**
Unit testing is a white-box testing technique performed at the code level or structural level. The purpose of unit testing is “to ensure that the code implemented the design properly” [3].

The author found 2 articles reporting automated unit testing. Laurie Williams et al. [49] have performed a case study at Microsoft. This case study consists of performing an automated unit testing using NUnit framework and a survey to get views about the experiences of developers, testers performing automated unit testing. Automated unit testing was performed on two versions of software. First version was of 1000 KLOC and second version of 1200 KLOC. Leonardo Mariani et al. [50] have proposed a framework called JAT (Jade Agent Testing Framework) to develop and then run test scenarios for MASs. For this purpose, authors developed a tool based on agent based platform named JADE. Three types of tests were run using this framework i.e. unit tests, integration tests and system tests.

• **Integration Testing**
The purpose of integration testing is “to verify that each software unit or module interfaces correctly to other software units or modules”. It has two approaches i.e. top-down integration and bottom-up integration [8].

The author found 3 articles reporting automated integration testing. Leonardo Mariani et al. [50] have proposed a multi-agent testing framework called JAT which can run integration tests (see section Unit Testing). Jean Hartmann et al. [60] have used the model-based approach for automated integration testing (see section Model-based Testing). Michael Wahler et al. [67] have presented a specification-based approach for automated integration testing (see section specification-based testing).

• **System Testing**
System testing is one of the black-box testing techniques. It ensures that the system as a whole fulfills the requirements of the customer / end user [81].

The author found 3 articles reporting automated system testing. Emil Börjesson and Robert Feldt [51] have performed a case study to evaluate two visual GUI testing tools for their test automation applicability for system level testing. Authors found that test automation for system level testing is applicable using visual GUI testing tools and result in cost and quality gains. Leonardo Mariani et al. [52] have presented a technique called AutoBlackTest for automated system testing. This technique is based on Q-learning concept. Test cases were generated using this technique and a commercial tool was used to execute these test cases. Leonardo Mariani et al. [50] have proposed a multi-agent testing framework called JAT which can run system tests (see section Unit Testing).
• **Acceptance Testing**

Acceptance testing tests the software where end users are involved in the testing process. The purpose of acceptance testing is "to ensure that end users are satisfied with the functionality and performance of the software system" [8].

The author found 2 articles reporting automated acceptance testing. Geir Kjetil Hanssen and Børge Haugset [53] have performed a survey in a firm that were using Fit (Framework for integrated tests) for automated acceptance testing. Four Developers were interviewed to get survey data about experience of using Fit tests for acceptance testing. To measure the efficiency of the automated type acceptance tests, a joint case study was conducted by Ericsson, Germany and one of its customers Mannesmann Mobilfunk, a German cellular network provider [54]. The testing was performed at a circuit switching subsystem of GSM system. For this purpose, a statistical usage testing platform was extended by Mannesmann Mobilfunk. The case study had three phases i.e. test platform extension, test case selection and implementation and test execution.

• **Model-Based Testing**

Model-based testing is one of the black-box testing techniques that use behavioral models of the application under test to automate the generation of test cases. The behavior of the application is modeled using state machines diagrams [55, 56]. An algorithm traverses through different states and generates different sequences called patterns that show the real behavior of the application. These patterns are then used to generate test cases. Model-based testing has been used to test different aspects of the application i.e. security testing [55, 56], GUI testing [57, 58], mutation testing [59], integration testing [60], functional testing [61, 64], random Testing [62, 63]. The main advantages of Model-based testing are its maintainability i.e. instead of maintaining the test cases, only models are updated and most of the bugs are discovered during modeling [55, 57].

The author found 10 articles reporting automated model-based testing. William H. Allen et al. [55] used model-based testing with the combination of fault injection technique for the security vulnerability testing i.e. stacks overflow vulnerabilities, in network protocol server implementation. Fault injection is a technique for testing an application with invalid data. Percy A. Pari Salas et al. [56] have presented a model-based approach for the security vulnerabilities testing. This model-based approach consists of three different models i.e. specification model shows the application behavior, implementation model applies the concept of fault injection and attacker model describes the intentions of the attacker. The three models together perform Test case generation.
Tommi Takala and Mika Katarain [57] have reported the experiences of using model-based testing for automated GUI testing of android applications. This experience was performed on a mobile application. Three main tasks of GUI testing were discussed i.e. modeling of mobile application, test generation and test execution. Ying Hau et al. [58] have reported the model-based testing for automated GUI testing. The authors developed an algorithm which generates the model, generates test cases from that model, translates test cases into test script and executes them. All the process is automated and for this purpose, they have implemented a system.

Robert Nilsson and Jeff Offutt [59] have used a model-based approach for automated mutation testing of timeliness for real time systems. "Timeliness is the ability for software to meet time constraints". A mutation-based framework was evaluated in a case study performed on a control system. This framework has three main activities: modeling of the real time system and mutation based test criteria selection, test case generation and test execution. Model checking technique was used for test case generation. Jean Hartmann et al. [60] have reported the model-based testing for the automated integration testing of components. The authors used the UML modeling tool called Rational Rose to model the system. Tests were design using the Test Specification Language (TSL) which is based on category-partition method. Two separate tools were used for test generation and execution.

Lars Frantzen et al. [61] have presented a model-based approach for the automated functional testing. A behavior model was created using symbolic transition system model. Test cases were generated and executed using a tool called Jambition. Muhammad Zohaib Iqbal et al. [62] have proposed a model-based approach for automated random testing of embedded real time systems. The authors have combined evolutionary algorithm and Adoptive random testing strategies to improve the fault detection effectiveness. Stefan Mohacsi and Johannes Wallner [63] have proposed a model-based approach for automated random testing. Task flow models were created using IDATG (Integrating Design and Test-Case Generation) method which consists of use case tasks that are test scenarios. Most frequently used sequences of test steps are called building blocks that can be part of a use case task. A hybrid algorithm is applied to these building blocks to randomly form test cases. A commercial tool was used to execute these test cases. Felix Lindlar et al. [64] have proposed a model-based approach for automated evolutionary functional testing. The authors combined the evolutionary algorithms with the Time Partition testing which is a model-based testing methodology for embedded systems to perform functional testing.
• **Specification-based Testing**
Specification-based testing, a kind of black-box testing, refers to the testing of a program / software based on its specifications for which it has been developed [82].

The author found 3 articles reporting automated specification-based testing. Sarfraz Khurshid and Darko Marinov [65] have reported a novel framework called TestEra for automated specification-based testing of Java programs. This tool tests the java program methods. The specifications are the pre and post conditions of the methods. Guoqing Xu and Zongyuan Yang [66] have presented a framework called JMLAutoTest for automated specification-based testing. This framework performs testing on program methods. It automatically generates nonisomorphic test cases, execute them and use the method’s post condition as an oracle. Michael Wahler et al. [67] have presented an approach called CAST (Computer-Aided Specification and Testing) that allows the test engineers to automate the integration tests based on specifications. This approach provides the test engineer to write test cases using a domain specific language TESLA, run them through test execution engine, and get the results.

• **Concurrency Testing**
Concurrency testing involves determining the system or application to handle concurrent resources and multiple users accessing the same data [2, 8].

The author found 1 article reporting automated concurrency testing. Kari Kähkönen et al. [68] have presented an approach which combines dynamic symbolic execution and unfoldings techniques for automated testing of multi-threaded programs. This approach was implemented in an algorithm.

• **Fault based testing**
Fault based testing is a type of testing where faults are seeded intentionally in a program and tests are written to reveal those faults [23]. Mutation testing is a kind of fault-based testing where we create mutants (copies) of the original program to inject faults. Different test cases are run to kill mutants [69].

The author found 5 articles reporting automated fault-based testing. Mayank Singh et al. [69] have presented an automated mutation testing tool called Cistron. This tool implements different mutation operators to handle faults for Aspect Oriented programs. Cistron uses the static analysis to automatically detect equivalent mutants for all fault types. B. F. Jones et al. [23] have discussed fault-based testing technique using GA (Genetic Algorithm). Test sets were created using GA and applied on a program seeded with faults. Kai Pan et al. [70] have proposed an approach called MutaGen to automatically generate tests for mutation testing of
database applications. MutaGen is based on existing framework called DBSyn. According to authors' approach, SQL query constraints were transformed into normal program code. Mutants were generated for both SQL queries and for transformed code. Query-mutant killing constraints were incorporated into transformed code and tests were generated to satisfy query-killing constraints. Yoonsik Cheon and Carmen Avila [46] have performed automated mutation testing using a tool called JET. This tool is actually a random testing tool but was also used for mutation testing. Robert Nilsson and Jeff Offutt [59] have used a model-based approach for automated mutation testing (see section model-based Testing).

- **Interoperability Testing**
  The purpose of interoperability testing is to test “the software to check if it can inter-operate with other software component, software’s or systems” [83].

  The author found 1 article reporting automated interoperability testing. Sarolta Dibuz and Peter Kremer [71] have presented a model called MAIT (Model for Automated Interoperability test). A framework was defined using this model and performed interoperability testing on IP-based protocol, called ROHC. This model consists of different components which perform different roles.

- **Load Testing**
  Load testing is the testing of “a system’s behavior under both normal and at peak conditions”. The purpose of load testing is to “identify the maximum operating capacity of an application as well as any bottlenecks and determine which element is causing degradation” [84].

  The author found 1 article reporting automated load testing. Chang-Sik Cho et al. [72] have proposed a scenario-based approach for the automated load testing of online games. The authors used the game description language and virtual game map to define the logic of the game. The load data is automatically generated as defined in game description language.

- **Performance Testing**
  The purpose of performance testing is to verify that the system or application meets its specific performance requirements. These requirements include speed, scalability and stability of the system [2, 8].

  The author found 1 article reporting automated performance testing. Tiantian Gao et al. [73] have proposed a reactivity-based testing framework for the performance testing of web applications. The proposed framework is based on the idea of retrieving user patterns from web logs and obtaining usage patterns by applying users' perspective metrics. Test cases were generated automatically by using genetic algorithms on usage patterns.
• **Stress Testing**
Stress testing verifies that the system behaves "gracefully" under stress and doesn't simply crash [2].

The author found 2 articles reporting automated stress testing. Wei Hoo Chong [74] has discussed an automated stress testing tool. This is actually a Record and PlayBack (RPB) tool that is being used in Motorola for the testing of cell phones. RPB is a platform specific and consists of different modules. Jian Zhang and S. C. Cheung [75] have presented a method to automatically generate test cases for stress testing of multimedia systems. In the paper, authors have performed the reachability analysis and constraint solving to generate test cases.

• **Memory Leakage Testing**
"Memory leakage is the phenomenon of permanent memory occupation that appears when a module allocates the memory without ever de-allocating it". If this happen continuously, the available memory size wears out and result in system failure. Testing this kind of phenomenon is called memory leakage testing [76].

The author found 1 article reporting automated memory leakage testing. Marcantonio Catelani et al. [76] have proposed a method for automated memory leakage testing. In the paper, authors used the dynamic pseudo randomization technique to generate test sequences that produce actual operating conditions and stress level to test the system. Memory leakage can be monitored by reserving some memory bytes for some specific process. If these bytes start to increase, it shows the leakage of memory.

**3.4. INDUSTRIAL CASE STUDIES**
Table 3.4 shows the Industrial case studies for each testing type. The author found 17 industrial case studies mentioned earlier. The highest number can be seen for model-based testing and less number for code coverage testing, security testing, GUI testing, regression testing, usability testing, concurrency testing, fault-based testing, load testing, performance and environment testing i.e. zero article. The author found 2 articles for each module and acceptance testing while 1 article for each functional, unit, integration, system, random, specification-based and interoperability Testing. From the table, it shows that these testing types are more relevant to the industry.
3.5. RETURN ON INVESTMENT (ROI) FOR TEST AUTOMATION

No article was found through this systematic review which can give information about ROI for automated testing. The author has explored other sources i.e. literature etc, which explain how to calculate ROI? The data about ROI will be presented in discussion part.
3.6. BENEFITS OF AUTOMATED TESTING
After thorough study of selected articles, the author has found following benefits for automated testing.

<table>
<thead>
<tr>
<th>Sr. #</th>
<th>Benefits</th>
<th>Article #</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fault Detection</td>
<td>[32,46,49,50,57]</td>
</tr>
<tr>
<td>2</td>
<td>Reduction in Cost</td>
<td>[31,32,74]</td>
</tr>
<tr>
<td>3</td>
<td>Reduction in Testing Time</td>
<td>[67,69,74]</td>
</tr>
<tr>
<td>4</td>
<td>Higher Test Coverage</td>
<td>[32,67,69]</td>
</tr>
<tr>
<td>5</td>
<td>Reduction in Test Effort</td>
<td>[31,57]</td>
</tr>
<tr>
<td>6</td>
<td>Repeatability</td>
<td>[67,74]</td>
</tr>
<tr>
<td>7</td>
<td>Increase Reliability</td>
<td>[32,74]</td>
</tr>
<tr>
<td>8</td>
<td>Improved Product Quality</td>
<td>[49]</td>
</tr>
<tr>
<td>9</td>
<td>Reuse of Tests</td>
<td>[74]</td>
</tr>
<tr>
<td>10</td>
<td>Reduction in Manual Error</td>
<td>[57]</td>
</tr>
</tbody>
</table>

Table 3.6: Benefits of automated Testing

❖ Fault Detection
Jason McDonald et al. [32] have described that one of the benefits of automated testing is fault revealing capability. Yoonsik Cheon and Carmen Avila [46] have used a random testing tool JET for mutation testing and the tool gave 90% fault detect rate. Laurie Williams et al. [49] have discussed the results of a case study at Microsoft about the experiences of developers and testers. The authors mentioned that the team of developers found 20.9% decrease in defects. Leonardo Mariani et al. [50] have proposed a framework called JAT to develop and then run test scenarios for MASs. The authors said that using JAT, test scenarios can be build which achieves high fault detection. Ying Hau et al. [57] has discussed one of the advantages of automated Model-based GUI testing is full fault detection capability.

❖ Reduction in Cost
Jason McDonald et al. [32] have mentioned benefits of automated testing in an industrial case study. The authors described that test automation reduces the cost of regression testing. Usha Santhanam [31] has described that automated testing minimize the cost of software due to repetitive nature of testing. Wei Hoo Chong [74] has listed advantages of using an automated stress testing tool called RPB. The author mentioned that few resources are needed to run the tests result in saving the cost.

❖ Reduction Testing Time
Mayank Singh et al. [69] have presented Cistron, a mutation testing tool. According to authors, they achieved higher test coverage and reduced testing time using this tool. Michael Wahler et al. [67] in a case study have mentioned that a complete manual test run took 3 to 5 days while
with the automated approach; it took just 15 to 30 minutes. Wei Hoo Chong [74] has listed advantages of using an automated stress testing tool called RPB. He mentioned that reducing total testing time results in shipping the product without delay.

- **Higher Test coverage**
  Jason McDonald et al. [32] have described one of the benefits of automated testing is higher test coverage. Mayank Singh et al. [69] have achieved higher test coverage using an automated mutation testing tool called Cistron. Michael Wahler et al. [67] in a case study have mentioned that automated testing gives increase in test coverage.

- **Reduction in test effort**
  Ying Hau et al. [57] has described that automated Model-based GUI testing reduces the amount of manual labor. Usha Santhanam [31] has mentioned that automated testing approach reduce testing effort several times over the manual approach.

- **Repeatability**
  Michael Wahler et al. [67] in a case study have mentioned that manual tests are not executed in the same sequence and often results are different with each run. Automated tests are executed with the same sequence and yield the same results providing 100% repeatability. Wei Hoo Chong [74] has listed advantages of using an automated stress testing tool called RPB. He mentioned that automated tests can be run repeatedly.

- **Increase Reliability**
  Jason McDonald et al. [32] have described that fault detection capability of automated testing and then rectification of these faults increases the reliability of applications. Wei Hoo Chong [74] has listed advantages of using an automated stress testing tool called RPB. The author mentioned that reducing the human involvement in testing decreases errors thus gives reliability.

- **Improved product Quality**
  Laurie Williams et al. [49] have discussed the results of a case study at Microsoft about the experiences of developers and testers. The authors mentioned that the team of developers found increase in product quality and they mentioned that this improvement is due to their interactions with the test team. The testers found that the code delivered from developers was of higher quality and it was hard for them to find defects.

- **Reuse of Tests**
  Wei Hoo Chong [74] has listed advantages of using an automated stress testing tool called RPB. The author mentioned that recorded test scripts eliminate the need to create the test script again, thus saving the testing time.
Reduction in manual error

Ying Hau et al. [57] have discussed one of the advantages of automated Model-based GUI testing is reduction in manual error.

3.7. EFFORT ESTIMATION

3.7.1. Manual Test Effort Estimation

Daniel Guerreiro e Silva et al. [78] have proposed an approach which is based on the team efficiency and developed an effort estimation model to estimate the execution effort for manual test cases. Before developing the actual model, authors have performed some steps which explain the approach thoroughly. The steps include effort data analysis, formulated hypothesis and accumulated efficiency. This accumulated efficiency is an effort metric which is sum of all the steps executed by a team for all test cases divided by the sum of execution time corresponding to these steps. After performing the above mentioned steps, authors have developed the model through 7 steps. The developed model is given below:

\[ T = (1 + r) \times \frac{S}{WE_A} \]

Where \( T \) is the estimated execution effort in time units, \( r \) is the relative Root-Mean-Square error, \( S \) is the number of test cases steps and \( WE_A \) is weighted arithmetic mean [78].

Eduardo Aranha and Paulo Borba [79] have presented a method to calculate the test execution effort estimation for manual testing. This method is based on execution points of a test case. Execution point is "a measure for the size and execution complexity of tests that is calculated based on test specifications". First, authors have calculated execution points for each test. Second, authors have presented two models i.e. Test Productivity-based model and Regression model, which estimate the execution effort for a given test suit using this measure. In [80], authors have presented the same model as well as discussed how to select best tests for execution when the resources are limited. It was explained through a graph that best tests are those that have high test coverage and less execution effort.

3.7.2. Automated Test Effort Estimation

Érika R. C. de Almeida et al. [77] have presented a method called Weighted Nageswaran method, or N-Weighted to estimate the test effort based on use cases. This method divides the use case scenarios into two types i.e. ones which have normal flows and the others which have exceptional flow. Exceptional scenarios are part of normal scenarios. This method first identifies the actors then weights them according to their interaction with system and gets the value called Unadjusted Actor weight (UAW) which is sum of all actors' weight. Second, it assigns weight to each use case based on a formula given blow:

\[ P_T = P_N \times N + P_E \times E \]
Where \( P_T \) is the use case's weight, \( P_N \) is the weight for normal scenarios, \( P_E \) is the weight for exceptional scenarios and \( N \) and \( E \) refers to the number of normal and exceptional scenarios. The sum of all use cases weight is called Unadjusted Use Case Weight (UUCW) [77].

Third it takes into account the technical and test environment factors, assigns them value according to their availability and degree of importance, multiply them to get the value called Technical Environment Factor (TEF). Forth, to get the final value called adjusted use case point (AUCP) by a formula given below:

\[
\text{AUCP} = \text{UUCP} \times (0.65 + 0.01 \times \text{TEF})
\]

Where \( \text{UUCP} = \text{UAW} + \text{UUCW} \).

The final effort is obtained by the formula give below:

\[
\text{Effort} = \text{AUCP} \times \frac{\text{man-hour}}{\text{use case point}}
\]

Where 3 man hours were suggested to plan, write and run tests per use case point. Plus 5% effort value is added for project complexity and 5% for coordination and management [77]. In the paper, authors compared this method with other three methods.

Eduardo Aranha and Paulo Borba [80] have discussed about the effort spent on automating the tests. The authors have explained through graph that the test engineers can decide which tests should be automated by considering the effort to automate and execute each test. The authors also have showed the intention to create a model for this purpose which is in progress.
4. DISCUSSIONS
This section presents discussions on the basis of knowledge and results obtained through this systematic review. In this systematic review, we have tried to find the answers of the above-mentioned questions (section 1.3).

In the background section, the need for automated testing, automated testing process in detail has been elaborated and it provides motivation for this review. A comprehensive method have developed to conduct this review which included systematic way of searching the research articles, quality assessment of articles, final selection of articles and extraction of data for result purposes. After thorough analysis of the articles, 60 research articles were selected and results have been presented in the form of statistical and descriptive form.

The purpose of this thesis was to conduct a systematic review and to find answers of the research questions. The answers have been found for all questions except return on investment for automated testing through this review. For the first question, 22 automated testing types were found which have been described in results section. Total 56 articles were found describing different automated testing types. Different approaches were used for automated testing types for different purposes. Statistical analysis shows that code coverage testing, model-based testing, functional testing, GUI testing are most favorite for researchers to investigate. It also shows that research in automated testing started to increase from 2006 and still continue. However, no article was found about code-based testing which was expected to be found. The reasons for not finding any article about code-based testing might be (1) lack of research work in this area (2) using of one database instead of adding more databases in search strategy. For the second question, 17 articles out of 56 articles were of case studies conducted in industry. The less number of case studies conducted in industry are due to lack of resources and it requires much time.

- Lack of research articles about ROI
There was no article found giving evidence about calculating the return of investment for automated testing. To find the answer of this question, an un-structured way of search was adopted i.e. literature search. The reasons for lack of research articles about ROI can be (-1-) mostly Software Organizations are interested to estimate/calculate ROI for their testing activities so there is lack of academic research on ROI (-2-) Software Organizations are not willing to publish their internal matters which may be beneficial for their competitors in Software Industry (-3-) every Organization has its own aims and objectives so its way of ROI calculation is different from other Organizations.
Return on Investment for automated testing

Dustin Elfriede et al. [2] have described in detail how to estimate the ROI for AST. For this purpose, authors have identified the four major activities of AST process and calculated the cost savings for each activity. The four activities are the following:

- Test Environment setup
- Test development
- Test execution
- Test evaluation

These major activities have further sub-activities. The ROI calculation were made in the form of worksheets in which time savings and cost savings for both manual testing and automated testing were calculated. Three types of time/effort calculation were made i.e. test development time, test execution time and test evaluation time. The estimated time for one automated test development is 50% more than manual test development. The execution time for one automated test is 20 times less than manual test execution time. The test evaluation time for automated test is 10 times less than manual test evaluation time. These time calculations are just estimation but it can vary depending upon the experiences of test engineers, the type of test cases. The authors have also mentioned other factors to consider when estimating ROI i.e. automated testing tools cost, cost of tool training if test engineers are not skilled in test automation, test case maintenance cost, risks etc [2].

Dorothy Graham and Mark Fewster [86] have described a case study for automated Model-based testing. In the case study, authors have explained the ROI activities for test automation. Test effort was compared for both manual and automated approaches. The activities, for which effort was calculated, were test creation, test automation framework preparation, test execution and test maintenance. For test creation, it took 45% approximately more time for automated tests than manual test creation. Test automation framework preparation includes the following activities:

- Testing tool acquisition, installation and configuration
- Tool training
- Testability Analysis
- Getting other information to create test model

The effort for above activities was calculated. The execution time for automated tests was approximately 11 times less than manual tests. Finally the maintenance time for automated tests was 9 times less than manual tests. It is worth to mention here that the tests maintenance
time for other types of testing may vary. Initially, the testing effort was increased for automated testing. After the fourth test run, it was approximately equal for both testing approaches. This is called breakeven point. A breakeven point is the point at which the testing time/effort for manual and automated are approximately equal. After the 20 cycles, the time for manual testing was 3735 hours and for model-based testing 1135 hours. One of the benefits of Model-based testing is that instead of maintaining the tests, only models are updated. [86].

Manual Testing effort = Creation + Execution + Maintenance [86]
Automated Testing Effort = Creation + Automation + Execution + Maintenance [86]

<table>
<thead>
<tr>
<th>Activity</th>
<th>Manual Hours</th>
<th>Model-based / Automated Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Creation</td>
<td>180 test cases @ 1 hr: 180 hr</td>
<td>Creation of 2 scenarios @80 hr: 160 hr</td>
</tr>
<tr>
<td></td>
<td>Manual test creation total: 180 hr</td>
<td>180 test case @1 hr: 180 hr</td>
</tr>
<tr>
<td>Preparation of framework</td>
<td>N/A</td>
<td>Acquisition and installation of tools: 60 hr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tool training: 40 hr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Testability analysis and improvement: 200 hr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Obtaining information for test model: 160 hr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Framework preparation (automation) total: 460 hr</td>
</tr>
<tr>
<td>Test Execution (per cycle)</td>
<td>180 test cases @ 45 min: 135 hr</td>
<td>180 test cases @ 4 min: 12 hr</td>
</tr>
<tr>
<td>Test maintenance (per cycle)</td>
<td>180 test cases @15 min: 45 hr</td>
<td>2 scenarios @ 2.5 hr: 5 hr</td>
</tr>
<tr>
<td>Total Effort</td>
<td>Manual Hours</td>
<td>Model-based/Automated Hours</td>
</tr>
<tr>
<td>1 test cycle</td>
<td>315 hr</td>
<td>812 hr</td>
</tr>
<tr>
<td>2 test cycles</td>
<td>495 hr</td>
<td>829 hr</td>
</tr>
<tr>
<td>3 test cycles</td>
<td>675 hr</td>
<td>846 hr</td>
</tr>
<tr>
<td>5 test cycles</td>
<td>855 hr</td>
<td>863 hr</td>
</tr>
<tr>
<td>5 test cycles</td>
<td>1035 hr</td>
<td>888 hr</td>
</tr>
<tr>
<td>10 test cycles</td>
<td>1935 hr</td>
<td>965 hr</td>
</tr>
<tr>
<td>20 test cycles</td>
<td>3735 hr</td>
<td>1135 hr</td>
</tr>
</tbody>
</table>

Table 3.5: ROI calculation [85]
For the fourth question, 4 articles were found providing evidence about effort estimation. Different models were presented and comparison was performed. However, there is a need for more research to develop more models for estimating effort of AST. The results of retrieved data have been presented. To present the more reliable results, there may need to involve more persons to conduct the review as well as more data sources can be added. The searching of articles related to research questions depends mainly on the selection of keywords for the query. If you have more generalized keywords then you may find many articles that are not of any interest for your review. The author found many articles through the query that were not providing data concerning research questions. It is one of disadvantages of systematic reviews.
5. SUMMARY OF FINDINGS
This section provides findings of systematic review as well as findings of un-structured search. The author couldn't find any article providing evidence about ROI for automated testing. So the findings of un-structured search are not part of this systematic review. Total 60 primary studies / articles have selected for this systematic review and data have been extracted related to research questions.

5.1. FINDINGS OF SYSTEMATIC REVIEW

- Automated Testing Types
  We have found 22 automated testing types through our systematic review.
  1. Code-Coverage Testing
  2. Module Testing
  3. Functional Testing
  4. Security Testing
  5. GUI Testing
  6. Regression Testing
  7. Random Testing
  8. Usability Testing
  9. Unit Testing
  10. Integration Testing
  11. System Testing
  12. Acceptance Testing
  13. Model-based Testing
  15. Concurrency Testing
  16. Fault-based Testing
  17. Interoperability Testing
  18. Load Testing
  19. Performance Testing
  20. Stress Testing
  21. Memory Leakage Testing
  22. Environment Testing

- Most of the work done in automated testing started from 2006 while not much work was done during 1998-2005.
- Most of the studies, we found, are about code-coverage testing and model-based testing while less studies are about Usability Testing, Concurrency Testing, Interoperability Testing, Load Testing, Performance Testing, Memory Leakage Testing and Environment Testing.
• Most of the studies are of experimental type followed by cases studies, experience and then survey studies respectively.

• Benefits of Automated Testing
  1. Reduction in Cost
  2. Reduce Testing Time
  3. Higher Test Coverage
  4. Reuse of Tests
  5. Reduction in Test Effort
  6. Improved Product Quality
  7. Fault Detection
  8. Repeatability
  9. Increase Reliability
  10. Reduction in Manual Error

• Most of the articles provide benefits of automated testing are fault detection, reduction in cost, reduction in time and higher test coverage.

• Effort estimation models for manual testing and automated testing.

5.2. FINDINGS FROM THE SEPARATE INVESTIGATION OF ROI

• Reduction in test execution effort is main reason for ROI. If the same tests are run in multiple cycles, then the cost of manual testing increases while the cost of automated testing decreases after a few cycles.

• Automated testing provides ROI if a good strategy is adopted. Identify your business needs and plan your strategy according to your needs. It depends on a number of factors i.e. whether you want to increase efficiency or reduce your cost etc.

• There is a lack of research work for ROI.
6. CONCLUSIONS & FUTURE WORK

Finally, it is concluded that the thesis have successfully highlighted important aspects of the data through detailed results analysis, discussions and summary of findings. It also provides any gaps in research work, reasons for lack of research. The author has made effort to answer the questions. This thesis has contributed more knowledge in research work of automated testing with different manners. The purpose of this thesis has successfully achieved and it fulfilled all the requirements presented in [15]. It is worth to mention here that systematic review is good mean of finding and analyzing research data about a topic, phenomena and area of interest. It also provides support to researchers for conducting and investigating more research if there is a gap or lack of research for specific topic, phenomena or area of interest.

The results of this systematic review show that there is much lack of research in different areas of automated testing. This thesis report can help the researchers to investigate further these areas. The areas are presented below.

We have found only 1 article concerning the Usability Testing, Concurrency Testing, Interoperability Testing, Load Testing, Performance Testing, Memory Leakage Testing, and Environment Testing. Researchers can explore these areas of testing more to contribute.

There is a lack of industrial case studies for automated testing types (see section 3.3). Researchers can conduct more case studies in industry to investigate these areas. We couldn't found any article related to return on investment for automated testing. This area of testing needs much research work. There were only 4 articles providing evidence about effort estimation. This area can also be the topic of research work for future.
REFERENCES

2. Dustin, Elfriede, Implementing Automated Software Testing: how to save time and lower costs while raising quality, Addison-Wesley, 2009


