A Mapping Study of Automation Support Tools for Unit Testing

Master Thesis
Author: Inderjeet Singh
Email: ish10001@student.mdh.se
IDT Supervisor: Adnan Causevic
IDT Examiner: Sasikumar Punnekkat

Mälardalen University
School of Innovation, Design and Engineering
June 2012
Sweden - Västerås

ABSTRACT

Unit testing is defined as a test activity usually performed by a developer for the purpose of demonstrating program functionality and meeting the requirements specification of module. Nowadays, unit testing is considered as an integral part in the software development cycle. However, performing unit testing by developers is still considered as a major concern because of the time and cost involved in it. Automation support for unit testing, in the form of various automation tools, could significantly lower the cost of performing unit testing phase as well as decrease the time developer involved in the actual testing. The problem is how to choose the most appropriate tool that will suit developer requirements consisting of cost involved, effort needed, level of automation provided, language support, etc. This research work presents results from a systematic literature review with the aim of finding all unit testing tools with an automation support. In the systematic literature review, we initially identified 1957 studies. After performing several removal stages, 112 primary studies were listed and 24 tools identified in total. Along with the list of tools, we also provide the categorization of all the tools found based on the programming language support, availability (License, Open source, Free), testing technique, level of effort required by developer to use tool, target domain, that we consider as good properties for a developer to make a decision on which tool to use. Additionally, we categorized type of error(s) found by some tools, which could be beneficial for a developer when looking at the tool's effectiveness. The main intent of this report is to aid developers in the process of choosing an appropriate unit testing tool from categorization table of available tools with automation unit testing support that ease this process significantly. This work could be beneficial for researchers considering to evaluate efficiency and effectiveness of each tool and use this information to eventually build a new tool with the same properties as several others.

Key Terms: Testing, Unit testing, Test date generation, systematic literature review, automatic unit testing

ACKNOWLEDGEMENT

All thanks are due to my parents and foremost for their countless blessing. Acknowledgment is due to Mälardalens högskola for supporting this research.

My unrestrained appreciation goes to my supervisor, Adnan Causevic, for all the help and support he has given me throughout the course of this work and on several other occasions. I simply cannot imagine how things would have proceeded without his help, support, and patience. I also wish to thank my thesis Examiner, Professor Sasikumar Punnekkat, for his help, support, and contributions.

I also acknowledge my many colleagues and friends as I had a pleasant, enjoyable and fruitful company with them.

Finally, I wish to express my gratitude to my family members for being patient with me and offering words of encouragements throughout my thesis work.

Inderjeet Singh

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

In general, software testing is the process of executing the system with the intention to find faults in it. It is a very important phase in software development and it comprises approximately 50% of total development cost and resources required [14]. The software testing has many different sub-phases in it, such as: unit testing, regression testing, acceptance testing, etc. All these phases have equal importance, but only recently, unit testing emerged as a vital part of software development process [1]. Unit testing is a method of finding bugs in an early stage of a system development and it can eventually help in reducing the cost and effort needed to fix the bugs later at integration or system level [1][43]. Therefore, unit testing is a method of testing a particular unit of a system when it is not integrated with other units. Unit could be considered as any smallest testable part of an application [2]. Generally, unit testing is a task performed by developers. Performing unit testing is sometimes considered as time-consuming task for developers and generally it is not on their high priority list [2]. Test driver is a program that runs automatically all unit test cases and as an end result provides a list with the failing/passing unit test cases [43]. However, software-testing process is still heavily based on the manual testing approach and it typically involves high efforts in the development of test driver for the particular unit under test [43]. The construction of the test driver normally puts more burdens on the developer to perform unit testing and development of test cases manually.

Having the pressure of covering the increasing number of lines of code to meet the quality standard in the market, developers and test engineers are trying out different techniques, such as unit testing, to sustain themselves in the competitive market. Lack of motivation is another potential reason for not performing unit testing in some organizations. It is highly recommended for a project management to create motivating environment before applying unit testing throughout the project [2]. The problems and issues explained above, in particular about unit testing, could be sorted out to a high extent if several process steps in unit testing could be performed automatically. There are many tools currently available for automating the unit testing process. These tools can provide an automated support at the unit level in many ways, for example by generating test cases, detecting known bugs in the code, etc. A few examples of the available tools are: “DART”, “JTest”, “EXE”, etc [19][1][2]. All available tools are different in many aspects like: programming language support, availability, output they produce. It is difficult for a developer and a project management to decide on tool to opt, as the efficiency and effectiveness of each tool is different. Efficiency and effectiveness of tool are two different things; efficiency of tool is related to execution time and robustness of tool, while the effectiveness of the tool depends on different factors such as type of errors it can find, type of test cases or output it can generate and quality of generated output. The effectiveness of a particular tool depends on many parameters, such as: “*types of bugs found*”, “*testing approach used*”, and “*type of output produce*” [73]. Therefore, this thesis work is an attempt to help developers in the process of choosing appropriate tool based on their requirements.

This thesis aims to find currently available unit-testing tools supporting automation by using a systematic literature review (SLR) process [3]. The SLR process is a good option to discover maximum available literature on the specific research area and it helps in reducing the chances of a biased or

limited result. The result of this thesis work is consisting of two parts: one providing a list of the available tools with proper citations and the second showing the categorization of tools based on different parameters like availability, language support, testing technique used, etc. In summary, this thesis work is comprised as a systematic literature review (SLR) of the relevant literature for the purpose of investigating currently available tools with an automated support for unit testing.

1.1. OBJECTIVE OF THE THESIS

Unit testing represents one phase in software testing which can make big difference in software testing of the system [14]. Out of several, one major advantage of unit testing is a detection of the defects in the very early stage of the application development, eventually decreasing the complexity of bugs found [1]. It also reduces the effort required to fix the bugs since they are detected earlier rather than in lateral stages of software development. However, two major problems faced in performing unit testing is time and effort required [43]. To reduce the time and effort for unit testing, many unit-testing tools are available in the market that can automate this task to some extent. The objective of this thesis work is concentrated towards identifying tools that can automate unit-testing process. In addition, this work provides the characterization of tools based on domain, language, availability, testing technique and developer interaction. This research work also explains the most common "*type of error(s)*" each tool can find though reading research papers.

1.2. PROBLEM STATEMENT

Developers might find problem in performing unit testing since it is somewhat difficult for them to manage both the development of a module and testing of it [14]. The use of automated unit testing tools has solved this problem to some extent and provides a convenient way to perform unit testing. To the best of our knowledge there is no work published listing the current available unit testing tools with automated support. The lack of comprehensive study and categorization of tools based on well establish criteria is also a problem for developers to select appropriate tool. Therefore, developer or organization does not have any concrete work to look for such tools and their properties. The main problem for developers in selecting appropriate tool can be solved to the extent by this thesis work. This thesis work would provide maximum research information specific to automation of unit testing that could help developers or organizations to find and to choose suitable tool. As a result of this thesis work, the list of all current available automated unit testing tools is provided together with explanation of some additional properties that can be helpful for developers or organizations.

1.3. ORGANISATION OF THESIS

The rest of thesis report is organized as follows. Chapter 2 explains existing related work on these research topics i.e. systematic literature review and automated unit testing. Chapter 3 includes the detailed explanation of systematic literature review that can be useful as the study material for others. Chapter 4 presents the major sub parts of opted SLR method for this research work with the detailed explanation of research tool used for this thesis work i.e Zotero [61] in section 4.3. Chapter 5 discusses about some algorithms for generation of test data, which is one of the core part in developing unit-

testing tools with automation support. Chapter 6 is the analysis of SLR results extracted from SLR method. Chapter 7 is about a discussion on the validity of SLR results with regard to potential threats and possible solutions. Chapter 8 is explaining the summary of performed SLR process for this research work. Chapter 9 explains the categorization of listed unit testing tool with automation support. This chapter gives the detail description about the formulated criteria for categorization of tools. Chapter 10 discusses the type of error(s) that each tool can find through reading papers followed by chapter 11 and 12 that are about the conclusion and future work that can be done on this thesis further.

2. RELATED WORK

Since 2004, systematic literature reviews have earned enough popularity among researchers in software engineering. Several researchers have been regularly applying it to increase the scope in different fields of research in software engineering field [5]. During our research process we came across quite a few papers in the field of software engineering that have used the systematic literature review [5][6][7][8]. Mostly in all the cited papers, the SLR method has followed almost all the basic steps that are present in general SLR process such as forming research questions, search strategy used, study selection criteria, data extraction and data analysis [3]. There is one exception in a paper by *He Zhang* and *Muhammad Ali Babar* [5] that has used another method to collect data by interviewing persons involved in that particular research work. Out of all the cited papers, we found two papers those were very close to our research work.

Zulfa and *Shaukat* [7] have performed the systematic literature review specifically in software testing. Paper by Shaukat [7] is a systematic review aimed at characterizing empirical studies designed to investigate search-based test data generation cost effectiveness. This paper has followed all the basic steps involved in SLR but slightly lacks in the qualitative analysis part; instead, it answers explicitly each research question defined before in SLR. Second paper by *Zulfa Zakaria et al* [6] is a systematic review on investigating unit-testing approaches for business process execution language (BPEL) that is very close to our research work. The systematic review criteria have limited only to 10 years and an exclusion criterion is quite short and general such as based on the title and abstract.

The scope and number of references covered in the mentioned two papers [6][7] were not very comprehensive,783 and 450 respectively. Our systematic review recovered 1957 results by using eight digital search data sources and applied to exclusion criteria. The scope covered in our research is much higher than the respective papers [6][7]. Our motivation to use systematic literature review is to cover as much literature as we can and provide the systematic unbiased and transparent result.

In addition, we encountered some research papers that have addressed similar research question to us such as [1][49][74]. In research work [1], Mads and Mikael have presented a detailed survey on automation of software testing by taking each testing phase at time. At the end they have made a comparison between three selected tools which are Agitar, Jtest, JCute [23][10][26]. In our opinion, this paper lacks in explaining the clear reason behind the selection of these three tools. Similar type of comparison is presented in [49] by Smeets and Simons. Authors have applied the selected tools i.e. Randoop, JWalk, Mujava [49] on open source named as JPacman and closely monitored the behavior of each tool. The same drawback related to defend the selection of tools was found in this paper as well. Jon [74] has performed the survey on various test data generation techniques for example static and dynamic, goal oriented and path oriented. This paper covers most common test data techniques with proper explanation with the help of examples.

In summary to related work, there is rigorous research going on in the field of automation of software testing. Many companies and research groups are trying to take automation of software testing to great success. The development of effective automatic test input generator and intelligent Oracle seem to be

biggest challenge. Various algorithms have invented and opted to build automatic test data generation such as taboo search, genetic algorithm, ant colony optimization and simulated annealing explained in detail in section 5. A research paper from Michael at al [69] has explained method to generate automatic test data input system with the use of genetic algorithm. Michael at al [69] illustrates some important problems related to test data generation and draw some connection with genetic algorithm. Paper [68] authored by Tracey has explained the involvement of taboo search in structural testing.

3. SYSTEMATIC LITERATURE REVIEW METHODOLOGY (SLR)

Systematic literature review (SLR) is one of the integral parts of this thesis work. As explained in the section 1, this research work is mainly about investigating unit testing tools with automation support using SLR process. One major reason we opted for SLR was to provide the results based on some well-defined process with concrete evidences. SLR has some general steps, which help others to reach on the same result [4]. SLR is a set of guidelines or principles to identify evaluate and interpret all available research literature relevant to the particular topic or questions defined on that topic [3]. The systematic review is considered as a secondary study while the independent studies contributing to respective research work are known as primary studies [3]. The one pivotal purpose of the SLR is to produce concrete research evidence by bringing same level of existing research literature. It helps in indentifying existing gaps in current research that can be suggested in particular area for future investigation [9]. A systematic review as defined by Cook et al [9] is "*the application of scientific strategies that limit bias by the systematic assembly, critical appraisal and synthesis of all relevant studies on a specific topic".* Therefore, as SLR is a well-defined methodology to carry out the research, that gives assurance towards unbiased results, although it fails to work against publication biasing at primary study level [3]. SLR is somewhat scientific and transparent approach that provides the "*audit trial of reviewer's decision, procedure and conclusion*". SLR's target stays in identifying all relevant studies by accessing their quality and providing a comprehensive summary of high quality literature available respective to the research work or research questions [4]. Every systematic review starts from defining a review protocol that contains well formed "*research questions*" and methods to perform a review. Based on the defined questions, the rigorous search begins to collect relevant literature or papers on it [4]. The selection of scientific database(s) (e.g. IEEExplore, Science Direct) is depending on the field of research area. Once the relevant literature have been collected from the defined scientific database(s), the next step is to check the eligibility based on the inclusion/exclusion criteria defined previously in the review protocol [3]. The result is further processed by using clear and empirical approach to reduce the biasing. Some examples of other types of inclusion/exclusion techniques are based on editorial, reader's letter, interviews and article summaries to define the eligibility of a literature. The following sections describe the general steps involved in any systematic review, which is based on the given guidelines published in [3].

3.1. RESEARCH QUESTION

Defining research question(s) is the first step and most important part of the systematic review. The formation of research question(s) should be focused on the main research work. Defining correct research question(s) is very critical in SLR, because other steps solely depend on this [3][4]. It should be formed in a way that would be relevant for both practitioner and researcher, for example, normally researchers are more diverted to find the undetermined faults in already developed techniques while practitioners give more emphasis on the techniques they need to adopt by considering particular situations. Sometimes, the research question(s) are too large and out of context, and to avoid such complications *Petticrew and Roberts* [3] have suggested one technique to frame the research

question(s) called as PICOC (population, intervention, comparison, outcome and context). Population focuses on a specific group like testers; developers, etc. based on the research area. Intervention deals with the specific tools, methodology, approach or similar that needs to be addressed in the research work. Comparison depends on the research work, if research work demands any comparison among tools, methodology, etc it is important to mention it in the question. Outcomes should be relevant to both practitioner and researcher. It should not provide outcome out of context. Context is to define whether the study is to be performed at an academic or an industrial setting. By following PICOC technique the quality of research question can be improved and it is recommended by several researchers [3][4][9].

3.2. REVIEW PROTOCOL

The review protocol deals with the methods and planning needed to perform a systematic literature review. It is necessary to define the review protocol, as it helps in avoiding the biasing from the researcher at the very beginning, because it is possible that the researcher will select the individual studies based on his/her expectations. Several elements as clearly described sections should be included in the protocol, such as the following [3][9]:

Background: The basic idea behind performing the survey
Research Question: Forming of research questions that are intended to deliver the research work.
Research Keywords: Based on the research question(s), keywords need to be extracted.
Study Selection Criteria: Deals with the inclusion/exclusion criteria for primary studies.
Study Selection Procedure: Describe how the defined selection criteria would be applied, for example number of people included in process.
Study Quality Assessment: The researcher should have a checklist to assess the primary study literature.
Data Extraction Strategy: This includes the steps that define how the references for primary study have been obtained.
Synthesis of the Extracted Data: This section is to identify the techniques for synthesis of the data from selected studies.

3.3. IDENTIFICATION OF PRIMARY STUDIES

Once the review protocol is finalized, a systematic review can start with identification of primary studies based on the defined research question. Identification of a primary study depends highly on extracted keywords and formation of queries. If needed, queries could be modified according to the search mechanisms of the selected scientific databases. The formation of queries relies mainly on the usage of logical operators like “OR” and “AND”. While collecting all the scientific literature studies, it is important to document the work consistently, as the information on data sources change very frequently. Types of information that could be documented are: data source used, query used on database, date the search was performed and years covered by query search [3][4].

3.4. DATA EXTRACTION

This part of the systematic literature review deals with searching of relevant data among primary literature collection. All the literature collected as primary study, undergo proper extraction phase and in order to meet the selection criteria defied in the research protocol. Selection criteria could be performed manually or by using some research tool. Purpose of the extraction phase is to bring more precision in the collected literatures. The references left after going through selection criteria normally represent the final set of papers that require thorough reading[3][4].

3.5. STUDY QUALITY ASSESSMENT

The quality assessment is related to the selection criteria used for obtaining primary studies. There are mainly three types of quality concepts defined, which are as follows [3][9]:

Bias: This is a dodgy technique of diverting the results systematically from the true set of results.

Internal validity: This is mostly related to the design of systematic review, as it shows the extent to which the design and conduct can prevent the review from systematic errors.

External validity: It is also referred as applicability or generalizability and defines the extent to which the studies are applicable in the broader research area.

3.6. DATA SYNTHESIS

Once the collected literature is precisely filtered through the selection criteria, it is time to summarize the results obtained from it. Data synthesis includes the literature study based on population, context, and outcomes. It is used to find and maintain the consistency between the results [3].

3.7. ADVANTAGES OF SLR

The sections above described the general steps involved in the SLR process. Some major advantages of SLR process are as follows [3]:

1. The well-formed methodology avoids the biasing in concluding results.
2. This approach can help to give the consistent results with evidence and help to find the issues about the phenomena.
3. This approach covers broader view of the research area.

4. RESEARCH METHOD

This thesis work comprises a systematic literature review (SLR) of relevant literatures on empirical studies for the purpose of investigating currently available tools with automated support at the unit testing level. The basic steps involved in the SLR process are already explained in the section 3. The approach we have chosen is very similar to the one defined in section 3, but in a way that is more sorted and manageable. SLR process consists of various steps like designing review protocol, defining the research questions, selection of keywords, identification of scientific database(s), search strategy, study selection criteria, etc. The following sections describe in detail the modified SLR steps we specifically defined in the context of our study based on the guidelines given in papers by Keele and Cook et al [3][9]. The main aim of adopted SLR process was to gather maximum primary literature, to our research area. The other steps of adopted SLR process are shown below:

1. Fundamentals of SLR Process
2. Performing SLR process
3. Information Retrieve & Storage (Zotero Tool)
4. Study Selection and Data Screening
5. Analysis of SLR Results
6. Validity of SLR Results

The pictorial representation of our adopted SLR process is shown in figure 1. It shows the selected steps with their sub parts in our SLR process from its beginning to the end. In figure 1, the hierarchy starts from fundamental of SLR process, which contains sub steps like forming research question(s) and then formulating keywords from question(s) that are explained in more detail in section 4.1. After keywords formulation, these keywords need to be applied on the selected research databases. The selected research databases for opted SLR process, explained in detail in section 4.2, are seven in total and all are related to the software engineering field. The literature has been collected from all the databases by designing respective query for each database specifically. This full process comes under the section performing SLR process. The literature was saved on the local hard drive with the help of research tool Zotero, that is explained more in detail in section 4.3 i,e. information retrieve & storage. This section is introducing this tool and explaining all aspects of this tool used in this research work. Study selection and data screening section performs exclusion criteria with the aim to remove irrelevant references from the collected literature. The exclusion criteria are explained in details in section 4.4. Once exclusion criteria have performed, analysis of SLR results starts which is about to read all the relevant literature to find unit testing tools with automation support. Section 6 explains the found results after reading papers and the discussion about them in detail. The last step in our SLR process is to check the validity of obtained results in SLR. Section 7 speaks in details about the potential threats to the validity of SLR results and the solution to overcome these threats.

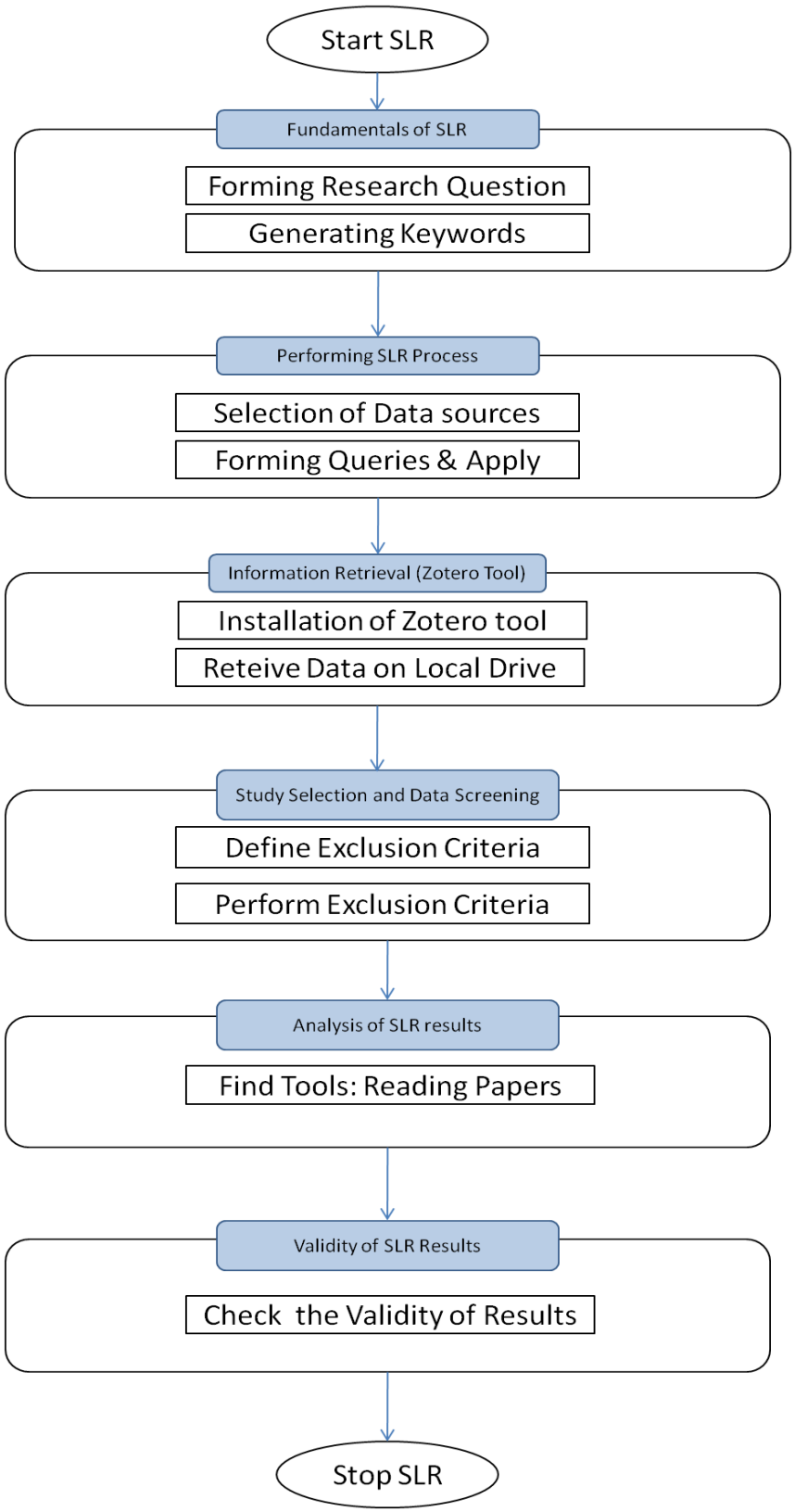

Figure 1: ***Systematic Literature Review Process***

4.1. FUNDAMENTALS OF SLR PROCESS

This section discusses establishing of the research questions and extraction of keywords out of it. The most important phase in SLR process is to define very accurately the research question(s) with the strong focus on the research area [3]. Therefore, our research work aims to search for the automation supported tools in unit testing and the question(s) should be concentrated to find such tools ultimately. While forming the research question(s) we found that our research would be concentrated to answer only one valid question, by targeting this question we will try to cover maximum primary literature. Therefore, by keeping PICOC [3] in our mind we concluded on this research question:

Can we identify and list currently available testing tools that can provide automation support during the unit-testing phase?

After the formation of such a research question, we need to define keywords based on it. Based on the question we extracted our main keywords that we will use on all the selected scientific database(s) listed in table 2. Generally, the reason behind forming keywords from question is to cover all aspects of the question and to maintain the consistency. The formulated keywords and the reasons to select them are explained in table 1 that is as follows:

Input Search Keyword	Reason(s) for Selecting Input Search Keyword
Test data generation	Test data generation is core element in developing testing tools with automation support, this keyword would help us to identify all such papers that are discussing this technique.
Automated unit testing	To target the unit phase of software testing with automation support,
Test generation tool	To identify tools that can generate the tests, so it covers two important things from our research question, first test generation and second is specifically finding “tools” for it.
Automated test case	To identify literature talking about generating automatic test cases at any phase of testing

Table 1: ***Keywords and Reasoning for Selection***

Table 1 explains the keywords with the reason(s) behind their selection, and it shows that the chosen keywords are covering wide scope in our research work. Defined keywords cover most of the important elements need to develop unit-testing tools with automation support such as test generation, automatic test generation, unit testing phase and so on. The keywords designed with close consultation with primary reviewer. It can be concluded form the table 1 that defined keywords are covering research question comprehensively.

After defining the keywords, the next integral part is to commence the searching of all relevant data on different scientific databases. The search keywords are more motivated towards two terms “Test generation” and “automated test”, even though the search question has more emphasis on unit testing.

The reason for not using “unit testing” explicitly in all keywords was to avoid limiting our search scope. Therefore, we made the combination of keywords in such a way that can result with maximum coverage in this field. The specific reason to select each keyword is explained in the table 1.

In this SLR process, in total three persons were involved, primary reviewer (Adnan Causevic) and secondary reviewer (Sasikumar Punnekkat), have reviewed the SLR process. One student (Inderjeet Singh) was in charge for performing SLR process. However, student took the initiative in all defined steps in SLR process under guidance of the senior reviewers. Completion of each step was reported to the primary reviewer for cross checking. Some key steps like forming keywords and search strategy were done by student under full consultation of primary reviewer. Other steps in SLR process were mainly performed by student such as including establishment of the research question, identification of scientific database(s) and study selection criteria. At the same time, senior reviewer (supervisor) was responsible for the work with monitoring and validating the review protocol, eligibility of selection criteria and search process itself. On several occasions, senior reviewer checked the implemented queries on some of the defined scientific databases for cross check of results and to minimize the random mistakes from the student.

4.2. PERFORMING SLR PROCESS

The commencement of our research started at looking at various high quality digital scientific databases listed below:

IEEE Xplore
ACM Digital Library
Academic search Elite
Google Scholar
Science Direct
Ingenta-Connect
Wiley web library
Springer Link

Table 2: ***Research Data Sources***

Search keywords were used individually on each research database. Our goal was to include maximum number of the relevant scientific databases related to our research area i.e. software engineering and computer science. Therefore, after researching a bit on the available options for databases, we selected databases based on the accessibility through the Mälardalens University library. Using specified keywords, queries were designed according to each database search capabilities. The use of queries on respective databases obtained a number of results from each database which were saved on the local hard drive with the help of Zotero tool [61]. The collected literature from each database was analyzed and the maximum number of results was obtained from Google scholar, which almost contained all the

result from ACM digital library and IEEE Xplore. The further analysis showed that the results obtained from Google scholar were not of very high quality, because it contained many references, which were not recoverable from provided link and many papers which were not written in English. The papers found in ACM digital library and IEEE Xplore were more centric towards the keywords we defined. The scientific database Wiley Web Library mostly gave the references for books on this research area. The query used on all the mentioned scientific sources was same with the logical operator "OR" between the keywords and with some minor modifications to accommodate specific search mechanism. The table 2 shows the query used on each data source with the number of results found:

Scientific data Sources	Query	Found results	Search Date
IEEE Xplore	TS= (("Test data generation") OR ("automated test case") OR ("automated unit testing") OR (" test generation tool")) where Metadata only = true	**382**	13th Feb 2012
ACM Digital library	TS= (("Test data generation") OR ("automated test case") OR ("automated unit testing") OR (" test generation tool")) where timeSpan= "all year" and Publication = "All";	**56**	13th Feb 2012
Academic search Elite	TS= (("Test data generation") OR ("automated test case") OR ("automated unit testing") OR (" test generation tool")) where Selected field = optional	**32**	13th Feb 2012
Google Scholar	TS= (("Test data generation") OR ("automated test case") OR ("automated unit testing") OR (" test generation tool")) where Search= title and abstract	**760**	14th Feb 2012
Science Direct	TS= (("Test data generation") OR ("automated test case") OR ("automated unit testing") OR (" test generation tool")) where Timespan= All Year And Publication type = all; In Expert search	**414**	14th Feb 2012
Ingenta Connect	TS= (("Test data generation") OR ("automated test case") OR ("automated unit testing") OR (" test generation tool")) where Search= "In article title, keyword or abstract"	**27**	14th Feb 2012
Wiley web library	TS= (("Test data generation") OR ("automated test case") OR ("automated unit testing") OR (" test generation tool")) where Search= " In all fields";	**201**	14th Feb 2012
SpringerLink	TS= (("Test data generation")OR("automated test case")OR("automated unit testing")OR(" test generation tool"))where Search="Title & Abstract";	**85**	13thFeb2012

Table 3: ***Queries on Each Data Source***

The total numbers of references collected from all the scientific databases by using queries were 1957. Out of which the maximum was gained from Google scholar database, 760 references. The collection of

literature was performed on two days i.e. *13th and 14th of February of 2012*. There were few references, from which no papers were obtained. Query designed for IEEE xplore was only targeting search filed as *"Metadata"* because with the option "*Full Text & Metadata*" was giving over 2000 results which were a lot by considering time allocated for this research work, so we decided to fix it on metadata. However, the results from IEEE Xplore did not have any limitation based on year and type of publications. Similarly for ACM Digital Library, we kept timespan and publication respectively to *"ALL YEAR"* and *"ALL"*. In Academic Search Elite, query was designed in such a way that can give results without any limitations of search fields such as *"metadata", "title", "abstract"* etc. This was achieved by leaving search field section to *"optional"*. Google Scholar query has search field to *"Title and Abstract"*, which eventually returned maximum references i.e. 760 out of all databases. The scientific database Science Direct has returned the second maximum number of results that were 414 with the query having *"ALL"* for both timespan and publication type. Query derived for Ingeta Connect database has set the search area field to *"In Article Title keyword and Abstract"*. This database found the minimum references i.e. 27, although there were not constraint related to year and publications. Wiley Web library has retuned 201 references with search area targeting *"In All Field"*. In the end, query for Springer Link generated 85 results targeting search area *"Title and Abstract"*. Consequently, it can be concluded that all queries for all databases were designed in a way to generate the results without any limitations on the date of publication and type of publications. Search area is different for almost all queries which were selected solely by considering the time constraints.

4.3. INFORMATION RETRIEVAL AND STORAGE (ZOTERO TOOl)

The study selection described in the section 4.2, is done using the tool called as Zotero [61]. Zotero is a tool for collecting, citing and sharing research purpose work. The installation and use of Zotero is quiet simple and is free available which focused on research work that automatically identify the contents to add or download directly to the hard drive [61]. The Zotero is available in different formats such as plug-in for browsers (Chrome, Firefox) and stand-alone application for system. Once you installed the plug-in in browser, it senses the content available on web page and offers to download the references from that page just by button click. The best part in the Zotero is that the downloadable contents can be saved on their web server as well, so there are no issues for losing data even the system crashes down. Zotero provides various features to make the work easier for researcher like creating folders, filtering papers based on some criteria by using "tag", "titles" and "*adding notes*" respective to each paper.

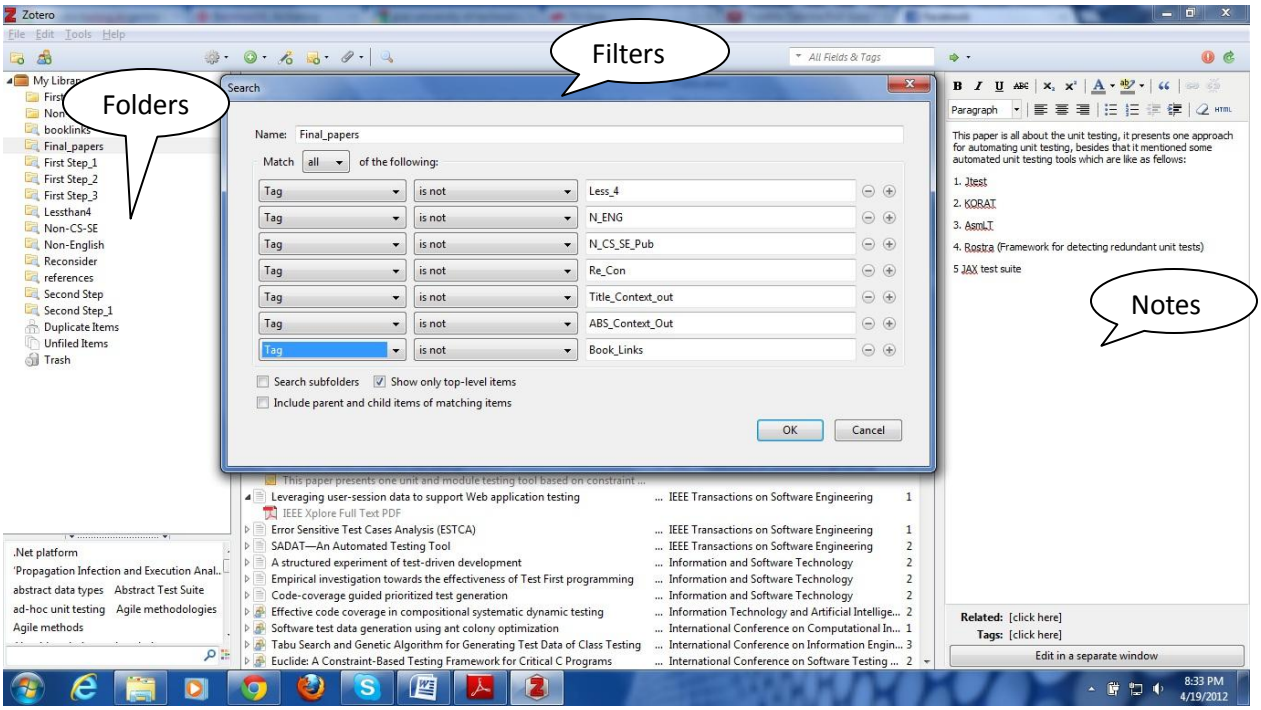

Figure 2: *Zotero Working Environment*

In this thesis work, we used both the plug-in version for chrome and stand-alone application. Zotero saved a lot of time to collect the paper but there were minor issues while collecting paper from all scientific databases, for example Zotero failed to collect papers from Springer link and also missed few papers from other scientific databases. The management of papers in Zotero is very easy and time saving. The figure 3 shows user interface of the stand-alone application.

The figure 3 shows, some features that we used during our research work, which are as follows:

1. Detection of duplicate references
2. Adding tags to separate reference from all other references
3. Addition of notes to add relevant information related to respective reference
4. Generation of bibliography with Zotero in doc file.

The study selection described in section 5, the exclusion criteria has applied with the use of "*tags*" in the Zotero. It helped a lot to filter papers systematically. The tags used for the filtering process was as follows:

TAG NAME	SIGNIFICANCE OF TAG
N_ENG	Paper which are not in English
Less_4	Papers contain less than 4 pages
Book_link	References for book, thesis
N_CS_SE_Pub	Publications do not belongs to computer science and software engineering
Re_Con	Papers to reconsider
Title_Context_Out	Paper's title not relevant to our research area
ABS_Context_Out	Paper's is not relevant to our research area after reading abstract

Table 4: ***Tags Used in Zotero for Filtering***

Zotero provides an option to add "tag" for each reference. Based on the added "tag" for each reference, it allows to create the new folder containing all the references with that "tag". All this process is automatic in the Zotero, which is very timesaving. After performing, the automatic filtering based on defined tags in table 6, the final papers that are 112 was downloaded on the hard drive and started reading each of them for finding the automated unit testing tools. However, reading papers in Zotero also helped in a way to add the notes related to each paper. The figure 3 shows the way to add the notes to the reference.

4.3.1. ZOTERO INSTALLATION AND USE

Zotero is new and still developing tool for research related work. It is very easy to install and configure on system even for new researchers. Zotero team provides very good documentation with separate forum to discuss current issues in Zotero. It is available for all three major operating system e.g. Linux, Mac and Windows. The basic step for installation and use of Zotero that was performed by us are defined below:

1. Download Zotero plug-in for chrome and standalone application from given link, http://www.zotero.org/support/3.0.
2. Register yourself on Zotero website to obtain user name and password. It will help you for synchronization with web server of Zotero and Zotero version on your system.
3. Open standalone application and go to tools->options…->Advanced, to assign the local directory on for system. This step is not mandatory.
4. In same window which was used in step 3, select sync option now and give your credential that you made while registration.
5. Now the standalone and plug-in application is sync with Zotero web server and it is ready to download content from web pages.

4.4. STUDY SELECTION AND DATA SCREENING

After collecting all the references in Zotero [61] explained in section 4.3. It was time to perform the selection process on all the references. In selection process, the exclusion criteria formulated include nine steps in total shown in table 5 below. Student under the proper guidance of primary reviewer performed the exclusion criteria.

Exclusion Criteria
Duplicate paper should be removed
Research paper should not be less than 4 pages
Exclude papers published in non-English language
Publications related to non-software engineering or computer science
Books and thesis work should be excluded
Title not appropriate to automation test data , testing and software tool should be excluded
Interviews, discussion should not be considered
Exclude papers not related to automation support of test data at unit testing level after reading abstract
Exclude Papers based on reading full text

Table 5: ***Exclusion Criteria***

Table 5 shows the exclusion criteria used to filter out the irrelevant literature from the collected data. In this process, some steps were easy to perform with the use of Zotero [61] like removing duplicates, book links and thesis work links because Zotero provides the inbuilt feature to detect the duplicates and book links. Except these few steps, other steps were quite time consuming because it was needed to go through from all the collected literature. The steps for removing the references based on the title, publication and abstract were the core part of the exclusion criteria. In particular, the step to filter papers based on title was bit complicated, because sometimes the title is not very informative and clear that creates confusion in deciding to include the paper or to exclude it. The step related to publication was also bit complex and time consuming, because of having some mixed publications of some software and non-software fields. It was tough to decide on some publication to opt it and needed to check the details about each doubted publication on other resources on internet. The final step in exclusion criteria is to select papers after reading full text, but that would be achieved only after removing papers based on abstract reading. The table 6 below shows the papers reduction hierarchy based on the exclusion criteria explained above in table 5.

Exclusion Criteria	Change in number of papers	
	Before	After
Remove Duplicates	1957	1545
Remove non- English papers	1545	1463
Remove papers less than 4 pages	1463	1400
Remove all the book links	1400	1248
Remove non CS_SE publication	1248	896
Remove papers based on title	896	378
Remove papers based on abstract	378	112
Remove papers based on full reading	112	42

Table 6: ***Summary of Applying Exclusion Criteria***

The total number of references collected from all scientific databases was 1957 including all types including duplicates copies, books chapters, non-English and interviews, etc. According to the exclusion criteria defined in table 5, student started filtering the papers in the tool Zotero [61]. In the collection, 412 papers found to have duplicate copies, after removal of duplicate papers; number came down to 1545 that had no duplicate copy of any reference. Out of 1545 paper, 63 references were not in the International language i.e. English, the reason for choosing English was not to limit our research to any specific language. Next step was to remove the references that have less than four pages in the article. Exclusion of such papers took the number down to 1400. After discussing with primary reviewer, we decided to remove all the references that include the books, bachelor or master thesis work and interviews. Removal of books and thesis work took the number to 1248. Then, we started researching at the publications for each reference, by inquiring all the publications; we removed all the references that were not related with software engineering, software testing, computer science and software verification and validation field. We found 378 references were not related to the publications mentioned above and number reduced to 896. Until here, the filtering was considered to be in phase one, because until this level we actually did not read any paper thoroughly. The final and second phase consist of filtering papers based on title ,reading abstract and reading full text that were involved reading the papers to the extent. Removal of paper after reading title took the number to 378. The abstract reading of 378 papers and exclude them were time consuming task, but it benefited later and number reduced to 112. This set of papers need full reading and then removing unrelated papers from them. The real focus would be on these papers to find the tools and approaches related to automated unit testing. During all this process, we found that some papers were not recoverable, either few were

not available for Mälardalens University library or paper were not available on that link. We put them aside in another folder named as "*Reconsider* *".

In the exclusion criteria explained above in table 5, we mentioned a step related to publication in exclusion criteria. There were many publications which were not related to our research work. So we decided to exclude them, due to the reason of having the high number of unrelated publications, we are showing the publications we included in our research work. The table 7 below shows the major publications we included in our research work.

Publication Name	Type of Publication	Number of References Collected
IEEE Transactions on Software Engineering	Journal	**29**
ACM SIGSOFT Software Engineering Notes	Annual Conference	**10**
Software Testing, Verification and Reliability	International Journal	**9**
International Conference on Software Engineering	International Conference	**9**
Software Testing, Verification, and Validation	International Conference	**5**
Information and Software Technology	Journal	**4**
Journal of Systems and Software	Journal	**4**
Software: Practice and Experience	International Journal	**3**
Theoretical Aspects of Software Engineering	Conference	**2**
Secure Software Integration and Reliability Improvement	Annual Conference	**2**
Quality Software	Journal	**2**
Others		**33**

Table 7: ***Important Publications Considered***

Table 7 above shows the major included publications with their type of publications in the research work. The major contributors in terms of number are ACM and IEEE. Type of journal field in table is to show whether the publication belongs to conference or to journal. Publications with the contribution of only 1 reference have added to the "*Other*" section, because the number was very high to show all the publications in the table 5.

*Recoverable folder is containing all the references of literature that were not available to be studied.

5. STATE OF ART ON TEST DATA GENERATION

This section is to create the basis for discussing the results in section “Analysis of results”. The results we found have strong connection with test data generation; most of the results use various test data generation algorithms to develop the tool. Test data generation is a method to find relevant set of data that can satisfy the testing cases [72]. As all the results found are testing tools with automation support, this test data generation becomes more important for such tools. There are many algorithms available to generate automatic test data, based on different techniques such as Taboo search, simulated annealing, function minimization methods and dynamic data flow analysis. All the found tools are either using directly or indirectly any of these algorithms as basis for the development.

Test data generation is the core part to build automatic testing tool [67]. Our research area is unit testing which is heavily similar to white box testing. The first step is to select the test adequacy criteria like branch coverage or statement. Followed by this, next is to search a set of test data that can satisfy selected test adequacy criteria. Generating test adequacy data manually is time consuming and effort intensive process [66][67]. This problem has attracted researchers and companies to build automatic test data generator. As a result, for this problem a few automatic test data generation techniques have been introduced. In the year of 1996 Ferguson and Korel [66] divided these techniques into three classes which are random test data generator, structured test data generator and goal oriented test data generation. This was considered as most appropriate classification related to test data generations [68].

Random test data generator is a simplest technique; it selects random inputs for test data specific to the program that could be integer, string, heap, etc [66][69]. For example, a function is using integer as argument, this technique could provide random number of integers to pass. Structured test data generator technique is considered as strongest among all [69]. It provides the specific test data based on the specific path. It is consequently resulted in better solutions but its bit tough to find the test data. Goal oriented is the technique similar to structured test data generator. Instead of random selection, it generates input for unspecific path, because it is much better to get the input for any path than any random input [66][67][68].

After this, the problem is to search set of inputs from specified domain all possible input data that can fit in for test adequacy criteria [66]. This is known as search problem for automatic software testing. In the beginning, the most of the test data generator were based on gradient descent algorithm, but it was considered as very inefficient and time consuming [71]. So to overcome this problem meta- heuristic search algorithm has introduced for developing test data generator. Few search algorithms defined were as follows:

1. Simulated annealing
2. Taboo Search
3. Genetic algorithm
4. Ant colony optimization

5.1. SIMULATED ANNEALING

Scott Kirkpatrick, C. Daniel Gelatt and Mario P. Vecchi invented it in the year of 1983.The idea was taken from the annealing and metallurgy, which has the concept of heating and then cooling the material to increase the size of its crystal that reduces the defects[62]. In simple words, it is a strategy for finding solution for combinatorial optimization problems. Simulated annealing is kind of protocol that has the property of iterative improvement, it does not stick in local optima. The necessary characteristics of SA are as follows:

1. Configuration model
2. Move Set
3. Cost function
4. Cooling Schedule

Configuration model is basically a model having all feasible configurations, where model is called as design space. This model should have all the legal design but it surly contains some base design as well. In the move set phase, the property of iterative improvement exists. This approach evaluates the current design for the improvements. Then in the design perturbations are applied randomly and it is called as move. So eventually it should reach at move set which contains all feasible design in the model. Later in third stage cost function, it evaluates the goodness of a design. This phase is actually for finding the effectiveness of the perturbations applied in previous step. This phase helps in covering all the alternative design and finds the best. The last phase is cooling schedule, which is to choose the best design. The designs with the decrease in the cost function at any level will be accepted and vice versa [62][63].

5.2. TABOO SEARCH

The taboo search is a simple and efficient approach for solving optimization problem to overcome from the limitations from linear search functions. It was invented by Fred W. Glover in the year of 1986 and 1989 [64]. It has proven to be very fruitful for the improvement in performance of techniques using memory structure [64]. There are some other methods also with memory less design such as semi-greedy heuristics, prominent genetic and simulated annealing [65]. In general, Taboo search starts by looking into local minima and fresh moves in one or more taboo lists, and then these moves are marked as taboo. This information is very useful in retracing the previous steps applied. One of the uniqueness in taboo search is not to prevent the repetition of applied moves rather to confirm that it is not missed. Sometimes in taboo search, some attractive moves are needed to be missed as to maintain the stagnation in searching process. At the end, the searching criteria stops when the condition is fulfill. There are a few general steps consist in the taboo search that are as follows [64][65]:

1. Defining a representation of possible solutions
2. Defining the neighborhood
3. Chose the objective to evaluate neighborhoods

4. Defining the taboo list with aspiration criteria and termination criteria.

These are the key issues that involve in the taboo search for searching.

5.3. GENETIC ALGORITHMS

John Holland invented genetic algorithms (GA) in the year 1960 at university of Michigan. GA gets the popularity very early and gain the interest of researchers very fast. As a result, lot of work has started towards GA regarding either the specific behavior of GA or application of GA targeting particular purpose [66]. Some of GA's application includes are automatic programming, optimization, machine learning and social systems. Generally the GA begins with the random collection of solutions, then with the recombination and mutation process the optimum solution extracted out of population. In the whole process, the first process is to evaluation of fitness of solution in existing population that can be acted as parent for next solution. In the evaluation many solutions are matched and the solution having high fitness values would be expected to be in it. After it, the parent generates the offspring's by recombining and mutation, which results in new population. There are some general steps that are as follows [66][67]:

1. Identify the minimization problem
2. Choose the genetic encoding to maintain the sequence among operations and their parameters.
3. Design the fitness of each population based on the problem
4. Check the iteration limit, if it does not reach upper limit, proceed next step otherwise repeat.
5. Apply the standard evaluation, crossover and mutation operator.

These steps of GA are valid for solving general optimization problem.

5.4. ANT COLONY OPTIMIZATION

Ant colony optimization (ACO) is a technique that was introduced by Marco Dorigo in 1992 in his PhD thesis. The idea has inspired from the behavior of ant colonies for finding the optimized path. It is very popular nowadays and included in many problems like train scheduling, time tabling and shape optimization, etc, but this approach has great importance in telecommunication, especially in routing and load balancing. The Ant colony optimization can be illustrated by simple algorithm that is explained below [70]:

```
Set parameters, initialize pheromone trails
SCHEDULE_ACTIVITIES
  ConstructAntSolutions
  DaemonActions    {optional}
  UpdatePheromones
END_SCHEDULE_ACTIVITIES
```

Figure 3: ***Algorithm for Ant Colony Optimization***

Therefore, to perform this algorithm, some notations are as follows

C, denoted for all the possible solution components and used to construct graph GC(V,E). The construct graph is associated with components and set of edges E.

A pheromone trails values t_{ij} would be associated to each C. Pheromone values defines the probability distribution of the solution that can be modeled. These values can also get update by ACO algorithm during the search.
So as it is shown in the figure 4, the meta heuristic started with the activation of three activities by the SCHEDULE_ACTIVITIES construct. This activity will be repeated until termination happens [70].

5.4.1. Construct ant Solution

In this step, it is mainly to construct the artificial ant solution from elements of finite set of solution components C. The process to construct the solution eventually leads to building graph GC (V,E). The choice of a solution would be done probabilistically at each construction level. The rules for probabilistic approach change on ACO variant [70].

5.4.2. DaemonActions

After the construction of solution, some problem specific actions need to apply which are known as Daemon actions. The most common daemon action includes the local search on the constructed solution, that local search is to find the pheromone value to be updated [70].

5.4.3. UpdatePheromones

The aim of this step is to keep the values of pheromones which are associated to good solution and while removes the pheromones values which are not related to bad solution. This can be achieved by two process one by decreasing the pheromone values which is called as pheromones evaporation and other by increasing the pheromones values [70].

6. ANALYSIS OF SLR RESULTS

This section evaluates the data collected from the systematic literature review process. The final list of papers collected through SLR process has aimed to find the unit testing tools with automation support available in the market. The table 8 contains the list of tools available in the market with the respective cited paper(s). This table is divided into three columns *tool name*, *primary citation* and *secondary citation*. The primary citation denotes that either the tool is introduced in the paper or the paper presents a detailed comparison of the tool. Secondary citation contains the papers that have very brief explanation of tool or have cited the tool name to draw some comparison. As mentioned in the table 8, the total number of tools found is 24, which includes all the domain, type of availability, etc. There are some tools mentioned in the table 8 that do not have any primary citation(s), those are normally commercial tools and are not freely available. In summary, it is found that 42 papers are talking about automated unit testing tools and are shown in the table 8.

Tool name	Primary citation	Secondary citation
Agitar		[23]
Austin	[26], [37]	
CAUT	[40]	
CREST(Successor of CUTE)		[26]
CUTE	[26], [37]	[11], [22], [30], [40], [41]
DART	[28]	[11], [26], [30], [41]
Eclat	[42]	[23]
EXE		[11], [41]
Findbugs		[16]
Jcrasher	[23]	[38], [43], [47]
Jcute		[26], [37]
Jtest		[10], 12],[18],[20], [23], [29],[34], [43], [46]

Tool name	Primary citation	Secondary citation
JUB(Junit test case builder)	[23]	
Jtst	[24], [51]	
Jwalk	[49]	
KLOVER	[19]	
Korat		[18]
PathCrawler	[41]	[11], [13], [22]
Palus	[60], [52]	
PEX		[22], [26], [38], [39], [41]
Randoop	[49]	
SMART (Extension of DART)		[11], [40]
SOATest		[20]
TestGen4j	[23]	[38]

Table 8: ***List of Unit Testing Tools with Automation Support***

Table 8 shows, the list of found results in alphabetically order, but this table does not give any more details about the tools. It contains all the tools irrespective of defining any boundaries such as Programming Language support or domain etc. It can be drawn that Jtest has referenced most of the time in the secondary citation column but it does not have any primary citation, one reason behind it could be that Jtest is a commercial tool that was developed by company Parasoft. Among non-commercial tools DART, CUTE and PEX are the three tools which found to have maximum citations. JCrasher is also quiet popular but it is now enhanced to new version called as Randoop. The output generated by each tool is also interesting point as not all the tools generate automatic test cases but the percentage of tools that can generate test cases is quite higher. There other ways to give output are like showing errors in the code line-by-line, providing code coverage etc.

6.1. PROTOTYPE TOOLS

During the search process for automation supported unit testing tools, we encountered some prototype tools developed by some educational group or researchers. In total, we found 11 prototype tools; most of them are based on the techniques explained in above section 5. The mentioned tools in the table 9, most of them are not available for public use; because these tools are not completely developed to launch in the market. The main purposes of these tools are to test the new techniques developed for test data generation. These prototype tools only involve the idea to build the new tool.

Prototype tool name	Primary citation	Secondary citation
AutoGen	[31]	
DiffGen	[15]	
Gozilla	[21], [48]	[23], [35],[36]
Juta	[25]	
SimC	[30]	
Slueth	[44], [50]	
Symclat		[23]
Symstra		[39],[40]
TAO	[32]	
TestGen	[45]	[10], [17]
Tgen	[33]	[14], [27]

Table 9: ***List of Prototype Tools***

In the table 9, it can be seen that almost all the tools have the primary citation field filled, which is somewhat necessary for prototype tools. Only two papers do not have the primary citation, but it could be possible that our keywords defined in section 4.1 did not match with it and failed to recover any paper. Among all the tools, Slueth and Godzilla are only tools with two primary citations, which show the popularity of these prototype tools. Prototype tool, TAO is based on the algorithm explained in section 5 i.e. genetic algorithms.

7. VALIDITY OF SLR RESULTS

This section explains the possible threats to the validity of SLR results in our research work. We identified mainly two potential threats to the validity of results in this research work, first was identified as the publication biasing in the collected literature. Second, was related to the quality check for the collected papers. In order to overcome from these threats, a well-defined review protocol has been developed containing research questions and extracted keywords out of research question(s). The keywords were formulated by close consultation of primary reviewer. To achieve the maximum results, queries have formalized under the proper consultation of primary reviewer and modified regularly as per their feedback. Primary reviewer was continuously crosschecking the usage of formed queries on respective databases that helped us in recognizing the random errors and the quality of literature. Review protocol included proper study selection and data extraction criteria in order to remove the irrelevant literature from the collected literature. Therefore, the precisely defined data extraction criteria played a big role in reducing the biasing at this level. Review protocol was well reviewed by both primary and secondary reviewers, which added more value in it and helped us in achieving maximum literature with minimum biasing. In addition, our strict exclusion criteria, played important factor in controlling the limits and not letting us to consider any irrelevant literature. The qualities of identified research papers were evaluated for their results. The quality of results is totally based on the relevant research papers obtained for this research work, so it was very important to perform quality check on the identified papers. The correctness of research papers were checked by considering many factors such as the authors, published in the publication organization (IEEE, ACM, SpringerLink), standard of conference or workshop, reputation of universities, research and development departments where the paper has been published. The removal of literature based on publications was important parameter in identify both the quality and biasing of the paper. As this criteria required us to study each selected publication organization in detail. In summary, the strictly performed SLR process, well defined question and keywords and continuous cross checking of process helped us in recognizing the biasing in publication. However, quality of collected paper has been assessed based on the criteria defined above.

8. SLR SUMMARIZATION

This section is about to summarize the SLR process we performed for our research work. Figure 4 shows all the opted steps from the beginning to the end in detail. Each big block in the figure represents each phase of our SLR process and the boxes inside showing some major task involved in each phase. The small box on each big box is showing the name of the SLR phase. Therefore, our SLR starts with defining research question and keywords, which comes under fundamental of SLR phase. It shows the selected keywords that are four in number. The queries have made out of keywords to apply on selected databases with some minor modifications in queries respective to each database. The figure 4 is also showing the number of references obtained from each database and it shows that maximum collected references are from Google Scholar (760). The process of selecting scientific databases and defining queries came under the phase performing SLR process. The collected literature from all eight databases has been saved on the local drive with the use of research tool named as Zotero. This process came under the information retrieval and storage, in total the number reached to 1957 and all references were collected by using Zotero. Once the collection of all reference is finished, the exclusion criteria have applied which is shown in the figure 4 in the phase *"study selection and data screening"*. There were eight steps in exclusion criteria and application of each step reduced the number of references. In last step, 112 papers were identified that were needed thorough reading. After the thorough reading of relevant papers, the "*analysis of results"* phase came which showed that we found 42 research papers discussing 24 automated support unit testing tools and 11 prototype tools. Once we found our results, it was important to perform the validity check on the result found and collected relevant research process. *"Validity of results*" was the last step of our SLR process, this step was about to identify potential possible threats to our derived results and the measures we took to overcome these threats. The core solution we found was to check the quality of publications such as author, recognization of research department etc.

It can be concluded from the discussion above and from figure 4, that the SLR opted for this research work was very simple and sorted out, so that anybody can understand and perform it to achieve same set of results. It was our one of the aim to design simple SLR methodology in order to achieve the highest quality result and succeeded to some extent.

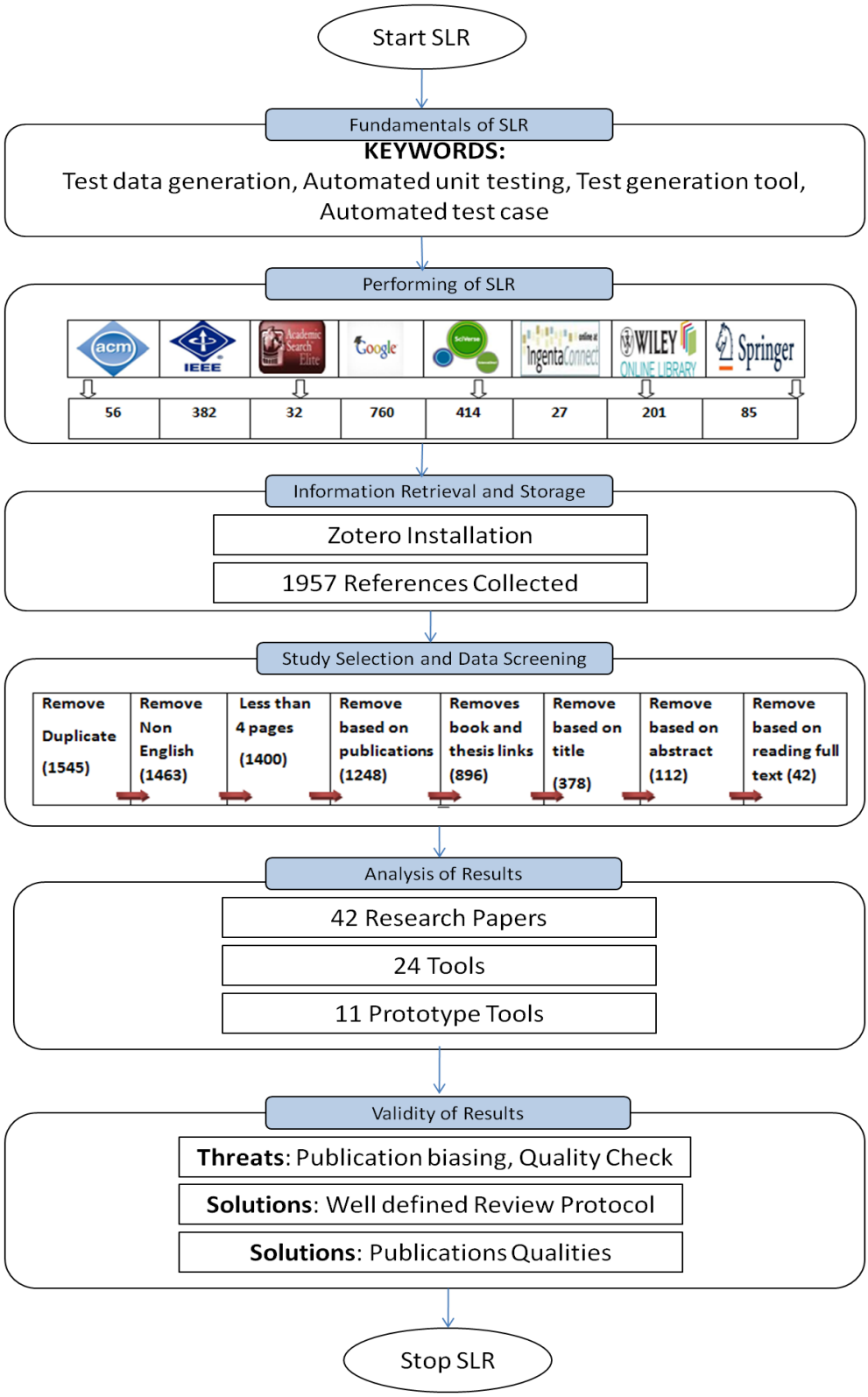

Figure 4: ***Detailed View of SLR***

9. CATEGORIZATION OF TOOLS

Once we collected all the automation supported unit-testing tools, shown in the table 8. It is time to categorize them and put them into groups. This is second phase of this research work and one of the most important one. Therefore, before starting the categorization of tools, it is important to formulate well-defined categorizing criteria. The categorizing criteria becomes more important to define on collected list of tools because this list contain all the tools regardless saying anything about their domain, language support etc. There are few points we decided to include in the categorizing criteria that are discussed below.

9.1.1. Based on Domain

This section categorizes the found tools based on the domain it belongs. This list collected in table 8 contains tools irrespective of any domain such as desktop, web or server. Accurately, domain defines as the type of application that is possible to test with the tool. The most common domains identified are Desktop, web based and server based. From table 11, most of the tools support only desktop-based applications. Only one tool is found to support web based services i.e. SOATest.

9.1.2. Programming Language Support

This part helps to categorize the tools based on the programming language supported. Each tool has the support for at least one specific programming language but few tools have the support for multiple languages. The most common programming languages concluded form the table 11 are Java and C. In the table 11, only one tool PEX has the support for Microsoft C#. All other's either support Java or C. CUTE is only tool with the support of both C/C++.

9.1.3. Testing Technique

There are many testing technique exist such as white-box, black box, path testing [71]. All these collected tools in table 8 are directly or indirectly based on any of them. This part will helps us to categorize the tools based on the testing techniques used by that particular tool. It is interesting to see in table 11, that tools are based on many different type of testing technique, there is not any one testing technique that has dominated the list of tools. Some testing techniques were found to be used by few tools such as "concolic testing" and symbolic execution.

9.1.4. Availability of Tools

The availability of the tools is also one of the important criteria to categorize tools. Availability of tool makes big difference for someone who wants to use. In the table 11, it can be seen that there are 3-4 commercial tools available such as Jtest, Agitar, etc. Most of the tools are open source but unfortunately, some of them are not even available. There are few examples of academic and licensed tools also like Eclat, PEX.

9.1.5. Developer Interaction

Developer interaction means the level of interaction needed by the developer to use that tool. This part actually has two important things to explain one is setting up environment for respective tool and other is the actual use of tool on module. The setting up is required for all the tools and the level of efforts required in this is different for each one. The setting up could take more time but it would be only one time effort. However, this section is more about latter part i.e. interaction. In the interaction part, developer needs to apply tool on the module. We decided to set the level on the scale of 0-2. Each value from 0-2 has some specific criteria defined, that is shown below in table 10:

Level on Scale	Type of interaction
"0"	Selecting unit or module in tool
"1"	Writing set of commands to use it on unit or module
"2"	Writing scripts for applying code on module

Table 10: *Developer Interaction*

Based on the criteria defined above, some fields are empty in the table 11, as the information found for these tools was not sufficient which is mainly because of availability of the tool.

Tool Name	Language supported	Testing Technique used	Domain belongs	Availability	Level of Developer Interaction
Agitar	Java	Observation driven testing	Desktop	Commercial	0
Austin	C	Search based technique	Desktop	Open source	
CUTE	C/C++	Concolic testing	Desktop	Free	1
CAUT	C	Symbolic and concrete execution	Desktop	Not available	
Crest	C	Systematic dynamic test generation	Desktop	Open source	

Tool Name	Language supported	Testing Technique used	Domain belongs	Availability	Level of Developer Interaction
DART	C	Concolic testing	Desktop	Open Source	2
Eclat	Java	Classification technique	Desktop	Academic	1
EXE	C	Concolic testing	Desktop	Open Source	2
Findsbugs	Java	Static analysis	Desktop	Open source	0
Jtest	Java	White box testing	Desktop and Server edition	Commercial from Parasoft	0
JCrasher	Java	Robustness testing or random testing	Desktop	Open source	1
JUB(Junit test case builder)	Java	Builder pattern	Desktop	Open source	1
Jtst	Java	Black box testing	Desktop	Not available	2
JWalk	Java	Specification based testing	Desktop	Academic	1
JCute	Java	Search based and Concolic testing	Desktop	Open source	0
KLOVER	C++	Symbolic execution	Desktop	Not available	1
Korat	Java	Constraint based generation	Desktop	Open source	
PathCrawler	C	Concolic testing	Desktop	Online Server	0
PEX	C#	Dynamically Symbolic exec	Desktop	License from Microsoft	1

Tool Name	Language supported	Testing Technique used	Domain belongs	Availability	Level of Developer Interaction
Palus	Java	dynamic and static program analysis techniques	Desktop	MIT License	1
Randoop	Java	Feedback-directed random testing	Desktop	Open source	1
SMART	C	Concolic testing		Open Source	
SOATest	Web Services		Web based applications	Commercial from Parasoft	0
TestGen4j	Java	Boundary level testing	Desktop	Not Available	

Table 11: ***Categorization of Tools***

10. TYPE OF ERRORS

This section deals with the type of error(s) that each tool can find. The aim was to find the type of error(s) for each tool by reading the research papers instead of by trying them on some source code. We successfully found the most common errors for 15 tools out of 24, only by reading the papers. The reason we could not find the type of errors for other nine tools was the lack of research papers available on them. The type of error(s) mentioned for each tool is not the only errors that they can find, these are some most common errors that they can find and regularly discussed in the respective research papers. Table 12 shows the type of error(s) we found for 15 tools.

Tool Name	Type of Error(s)
Agitar [56]	1. Error in mathematical calculation even inside nested statements 2. Unnecessary calls of same method 3. Results comparison with expected values 4. Null point exception and out of bound in array 5. Unused code Specialized rules (J2EE) 6. Formatting in the code
CUTE [26][53]	1. Unbounded loops 2. Constraints involving memory location 3. Transform non- linear expression 4. Pointer aliasing constraints
CAUT [40][60]	1. Array out of bound 2. Pointer aliasing and constraints 3. Static analysis and run time exceptions
DART [28]	1. Program crashes 2. Assertion violation 3. Non termination 4. Memory allocation detect with collaboration with other run time checking tool
EXE [11]	1. Constraint Solving 2. Independent constraint optimization 3. Bit-victor arithmetic

	4. Tracking indirect memory symbolically
Eclat [56]	1. Out of bound Exception 2. Pre/post condition violation 3. Illegal inputs
JCrasher [23]	1. Robustness failure 2. Undeclared run time exception 3. Pre/Post condition violation Out of memory error 4. Analysis of exception for error
JCute [37]	1. Deadlocks 2. Uncaught Exceptions 3. Infinite loop
JWalk [49]	1. Unexpected interaction among methods such as dead ends on the fly, broken pre condition 2. Interleaved constructor
Korat [18]	1. Violating pre conditions 2. Assertion to the problem of test generation for particular statement
Klover [19]	1. Out of bound in array 2. Infinite loop 3. Run- time exception
Path Crawler [41]	1. Infinite loop detection 2. Pre/Post condition violation 3. Handling floating point numbers and arithmetic operations 4. Aliasing and pointer arithmetic
PEX [53][54][55]	1. Argument exceptions 2. Out of bound in array 3. Arithmetic specification 4. Missing pre/Post conditions 5. Buffer overflow or Resource leak 6. Syntactic programming error

	7. Violation of assertion 8. Exhaustion of memory
Palus [59]	1. Run time exception 2. Out of bound error 3. Empty collection 4. Checking type compatibility before casting 5. Null pointer error
Randoop [49][58]	1. Run time assertion violation 2. Runtime access violation 3. Missing messages in resource file and state of resource file 4. Memory management errors 5. Concurrency error such that related to test input

Table 12: ***Types of Errors***

From table 12, it is visible that the most common errors that can find by maximum tools are "out of bound" and "exception related errors". The pre/post violation is another common error that most tools can find. PEX is the tool that can find maximum number of errors i.e. eight. It is interesting to notice that many tools are targeting memory related errors that shows the quality of the tools as it is always difficult to find the errors related to memory usage and blocks. In summary, it can be concluded that the tools are covering wide domain of errors.

10.1 DESCRIPTIVE ANALYSIS

This section is related to provide the descriptive analysis of collected and categorized data above in section 8 and 9. Table 13 shows, the grouping of tools based on the support to the programming languages. It is visible from table 13, Java programming language is supported by maximum number of tools i.e. 13 and then followed by C language i.e. 8. It can be concluded from the table, that the interest of developer for test data generation tools is more tilted towards the open source languages such as Java and C.

Language Support by tool	Number of tools
Java	13
C Language	8
C++ language	2
Web Services	1
C#	1

Table 13: ***Categorization of Tools based on Programming Language***

The table 14 shows interesting combination between the types of errors and number of tools. We made this table to find the most common error(s) that can be found by highest number of tools. It is interested to see that *exception related errors* are covered by seven tools out of 15 which is good percentage. At the same time, *unused code* and *check for program crashes* seem to be at least priority by the tool developers. Whereas, the assertion violation and constraint solving are also seem to be very important that is targeted by six tools. Assertion violation is a predicate like true/false statements. The constraint solving is kind of relation between variable values. By considering the total number of tools with the support of C/C++ language, the errors related to null pointer and pointer aliasing also look common.

Type of Errors		Number of tools	Percentage (%)
Infinite Loops		5	33%
Pre/Post Conditions Violations		5	33%
Program Crashes		2	13%
Assertion violation and constraint solving		6	40%
Memory Related Errors	Memory allocation defects	5	33%
	Indirect memory symbolically		
	Memory leaks		
	Exhaustion of memory		
	Out of memory		

Out of bound and null point		6	40%
Exceptions	Run time exceptions	7	46%
	Uncaught exception		
	Analysis of exception or error		
	Undeclared run time exception		
	Argument exceptions		
Unused code		2	13%
Pointer aliasing and null pointer error		4	28%
Illegal Inputs		3	20%

Table 14: ***Categorization Based on Type of Errors***

The figure 14 shows the summary for the categorization of tools based on the type of errors. The blue bars denotes the number of tools on left part of Y-axis and red line dot graph shows the percentage of tools out of total on the right side of Y- axis. The X-axis denotes the type of errors extracted from the table 12.

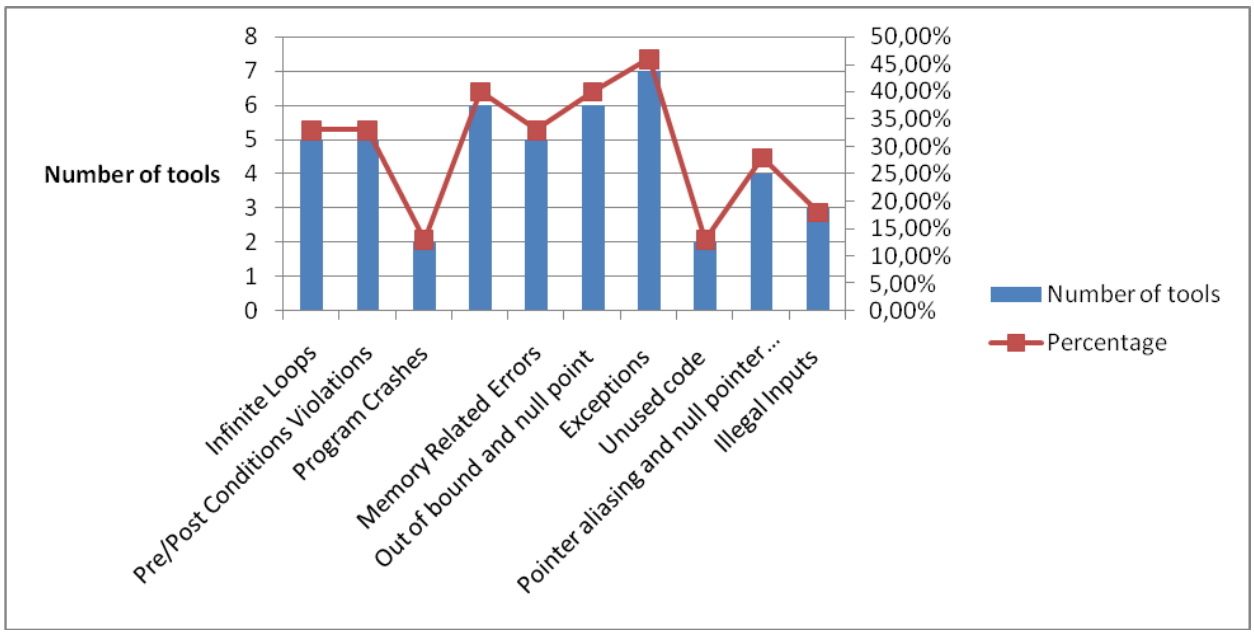

Figure 5: ***Graphical Representation of Categorization of tools Based on Type of Errors***

11. CONCLUSIONS

This section presents a summary of our major contributions in this research work to the software testing fraternity. This research work mainly contributes to unit testing phase in software testing. The problem to automate the unit testing is still a big concern in the software testing community. As per our best knowledge, there is no work published so far, which provides the list of such tools to refer. To solve this problem, we used the systematic literature review process to find the unit testing tools with automated support, through this process we collected all the distributed papers related to this field regardless the year of publication of papers. We collected 1957 primary studies paper and found 24 tools after applying the well-defined exclusion criteria on the collected papers. List contains the tools of all kind and domain such as different programming language support, domain, testing techniques, developer interaction and availability. This list of tools and categorization table could be very useful for someone who wants to choose automated unit testing tool. We also provide the list of prototype tools i.e. that are either underdeveloped or presents the idea to build tool.

The other major contributions of this research work are the categorization of all found tools and it also explains the type of error(s) that can be found by some tools. We categorized the tools based on five criteria that are Language support, availability, domain, testing technique used and level of effort needed to use the tool. The table 11 for categorization would be very useful in selecting the tool for industrial use. The table 12 tries to explain the type of error(s) that can be found by some tools. It gives more deep knowledge related to tools and to judge their effectiveness. We grouped the tools based on language and type of error(s), which is summary of the tools.

This research work contains the well defined systematic literature review process, which can be very useful for someone who wants to apply SLR in another research work. The SLR process has already explained in detail and with the support of another research tool named as Zotero in section 6. To conclude this research work, here is the summary of research question(s) and respective answers:

RQ1: Can we identify and list currently available testing tools that can provide automation support during the unit testing phase?

A1: Yes, we have identified successfully almost all tools that can provide automation support at unit testing phase. We have used well-defined process to identify tools and to our understanding, we successfully followed each defined step correctly. In total, we gathered 24 tools and eight prototype tools that can provide automation support at unit testing. The list of found tools and prototype tools are available in tables 8 and 9 respectively with proper citation.

12. LIMITATIONS AND FUTURE WORK

This section talks about the limitations and possibilities of taking this research work to next step. Due to the time and access to some resources, we encountered some limitations in our work. The following are the limitations in the research work:

1. The detailed information related to each found tool. Currently we only provide the categorization of tools; the next step could be to take each tool explicitly and explain the detailed information of each one.

2. This list contains some commercial tools, because of accessibility problem to these tools we could not provide the detailed information about them and especially type of error(s) that they can find.

3. The aim to provide the detailed comparison of some selected tools by applying them on open source could not achieved because of time constraint.

4. The idea to develop new tool by merging two three tools of similar kind that can find more range of errors.

Future work will try to address the above limitations, moreover for point one our research work is good platform to start because there is no need to apply rigorous search and categorization for tools. The categorization table i.e. 12 would be very helpful for the future researcher to look in and get the high-level idea about the each tool. Second limitation has great possibilities to be good research topic; some good-looking commercial tools such as agitar, Jtest and so on that deserve more research. One possibility is to ask for trial version of each tool from respective company so that student can perform all tests and utilize it to maximum. For third limitation, there are some key points that need to be considered such as:

1. Selection of tools by defining specific criteria, most appropriate would be based on Language. Student can take categorization table i.e. 12 as a reference.

2. Select appropriate open source to apply tools. Few suggestion are for “Java”:- Ant, Javaassist, for “C”:- VLC music player, Harbour etc.

3. Draw detailed comparison among selected tools by giving emphasis on type of error(s).

We tried to investigate point four also and found that this work is possible to achieve. This thesis work provides very good platform to group the tools based on types of errors and start with developing the new efficient and effective tool. Therefore, these are some possibilities with brief description to take this research work to next level.

13. BIBLIOGRAPHY

[1] Mads Bach-Sørensen, Mikael Malm, "Automated Unit Testing, A Survey of Tools and Techniques," in Department of Computer Science, Aalborg University 2007. Master thesis, Proceedings, Chapter 1.

[2] Rodney Parkin, "Software Unit Testing," in IV & V Australia: The Independent Software Testing Specialist , *1997.*

[3] Keele staff, "Guidelines for performing Systematic Literature Reviews in Engineering," in EBSE Technical report, *9th July 2007.*

[4] Hamid Riaz & Nidesh Tyagi, "Component- Models Classification: Life cycle dimension: A Systematic review," Master Thesis in School of innovation, Design and Engineering, Mälardalens högskola, *2012*.

[5] Muhammad Ali Babar and He Zheng,"Systematic Literature Reviews in software engineering: Preliminary Results from Interviews with Researchers", in Third International Symposium on Empirical Software Engineering and Measurement, IEEE, 2009.

[6] Zulfa Zakaria, Rodziah Atan, Azim Abdul Ghani , Nor Fazlida Mohd. Sani, "Unit Testing Approaches for BPEL: A Systematic Review," in 16th Asia-Pacific Software Engineering Conference, *2009.*

[7] Shaukat Ali, Lionel C. Briand, Hadi Hemmati and Rajwinder K . Panesar , "A Systematic Review of the Application of Empirical Investigation of search based test case generation" in IEEE Transactions on Software Engineering, *2010.*

[8] Emilia Mendes, "A Systematic Review of Web Engineering Research" in IEEE International Symposium on Empirical Software Engineering, 2005

[9] Cook DJ, Sackett DL and Spitzer WO "Methodlogies Guidelines for Systematic Reviews of Randomized Control Trails in Health Care from Potsdam Consultation on Meta- Analysis," J Clin Epidemiol 1995;45: 167-171.

[10] S. Khor and P. Grogono, "Using a genetic algorithm and formal concept analysis to generate branch coverage test data automatically," in *19th International Conference on Automated Software Engineering, 2004,* pp. 346–349.

[11] P. Mouy, B. Marre, N. Williams, and P. Le Gall, "Generation of All-Paths Unit Test with Function Calls," in *2008 1st International Conference on Software Testing, Verification, and Validation*, pp. 32–41.

[12] Yao Chen, Wei-Zheng Cai, and Yu Zhang, "The research and implementation of automatic unit test recording framework," in *2010 2nd International Conference on Software Technology and Engineering*, pp. V2–395–V2–399.

[13] M. Papadakis and N. Malevris, "A Symbolic Execution Tool Based on the Elimination of Infeasible Paths," in *2010 Fifth International Conference on Software Engineering Advances (ICSEA)*, pp. 435–440.

[14] S. H. Aljahdali, A. S. Ghiduk, and M. El-Telbany, "The limitations of genetic algorithms in software testing," in *2010 IEEE/ACS International Conference on Computer Systems and Applications (AICCSA)*, pp. 1–7.

[15] K. Taneja and Tao Xie, “DiffGen: Automated Regression Unit-Test Generation,” in *23rd IEEE/ACM International Conference on Automated Software Engineering, 2008*, pp. 407–410.

[16] Xun Yuan and A. M. Memon, “Using GUI Run-Time State as Feedback to Generate Test Cases,” in *29th International Conference on Software Engineering, 2007*, pp. 396–405.

[17] B. Korel, “Automated test data generation for programs with procedures,” in *ACM SIGSOFT Software Engineering Notes*, 1996, vol. 21, pp. 209–215.

[18] T. Xie and D. Notkin, “Tool-assisted unit-test generation and selection based on operational abstractions,” *Automated Software Engineering*, vol. 13, no. 3, pp. 345–371, 2006.

[19] G. Li, I. Ghosh, and S. Rajan, “KLOVER: a symbolic execution and automatic test generation tool for C++ programs,” in *Computer Aided Verification*, 2011, pp. 609–615.

[20] H. M. Sneed and Shihong Huang, “WSDLTest - A Tool for Testing Web Services,” in *Eighth IEEE International Symposium on Web Site Evolution, 2006,* pp. 14–21.

[21] R. A. DeMilli and A. J. Offutt, “Constraint-based automatic test data generation,” *IEEE Transactions on Software Engineering*, vol. 17, no. 9, pp. 900–910, Sep. 1991.

[22] A. Gotlieb, “Euclide: A Constraint-Based Testing Framework for Critical C Programs,” in *International Conference on Software Testing Verification and Validation, 2009*, pp. 151–160.

[23] Shuang Wang and J. Offutt, “Comparison of Unit-Level Automated Test Generation Tools,” in *International Conference on Software Testing, Verification and Validation Workshops, 2009*, pp. 210–219.

[24] K. Z. Zamli, N. A. M. Isa, M. F. J. Klaib, and S. N. Azizan, “A tool for automated test data generation (and execution) based on combinatorial approach,” *International Journal of Software Engineering and Its Applications*, vol. 1, no. 1, pp. 19–35, 2007.

[25] Y. J. A. Guo, “JUTA: An automated unit testing framework for Java,” *Jisuanji Yanjiu yu FazhanComputer Research and Development*, vol. 47, no. 10, pp. 1840–1848, 2010.

[26] K. Lakhotia, P. McMinn, and M. Harman, “An empirical investigation into branch coverage for C programs using CUTE and AUSTIN,” *Journal of Systems and Software*, vol. 83, no. 12, pp. 2379–2391, Dec. 2010.

[27] A. A. Sofokleous and A. S. Andreou, “Automatic, evolutionary test data generation for dynamic software testing,” *Journal of Systems and Software*, vol. 81, no. 11, pp. 1883–1898, 2008.

[28] A. Chakrabarti and P. Godefroid, “Software partitioning for effective automated unit testing,” in *Proceedings of the 6th ACM & IEEE International conference on Embedded software*, 2006, pp. 262–271.

[29] M. Harman, F. Islam, T. Xie, and S. Wappler, “Automated test data generation for aspect-oriented programs,” in *Proceedings of the 8th ACM international conference on Aspect-oriented software development*, New York, NY, USA, 2009, pp. 185–196.

[30] Z. Xu and J. Zhang, “A test data generation tool for unit testing of C programs,” in *Quality Software, QSIC 2006. Sixth International Conference on*, pp. 107–116.

[31] P. Bokil, P. Darke, U. Shrotri, and R. Venkatesh, “Automatic test data generation for c programs,” in *Secure Software Integration and Reliability Improvement, 2009. Third IEEE International Conference on*, pp. 359–368.

[32] S. Lapierre, E. Merlo, G. Savard, G. Antoniol, R. Fiutem, and P. Tonelia, "Automatic unit test data generation using mixed-integer linear programming and execution trees," in *Software Maintenance, 1999, Proceedings. IEEE International Conference on*, pp. 189–198.

[33] R. P. Pargas, M. J. Harrold, and R. R. Peck, "Test-data generation using genetic algorithms," *Software Testing and Verification Reliability.*, vol. 9, no. 4, pp. 263–282, 1999.

[34] M. Polo, S. Tendero, and M. Piattini, "Integrating techniques and tools for testing automation," *Software Testing and Verification Reliability.*, vol. 17, no. 1, pp. 3–39, 2007.

[35] A. J. Offutt, Z. Jin, and J. Pan, "The dynamic domain reduction procedure for test data generation," *Software-Practice and Experience*, vol. 29, no. 2, pp. 167–194, 1999.

[36] A. J. Offutt, "A practical system for mutation testing: help for the common programmer," in *Test Conference, 1994. Proceedings., International*, pp. 824–830.

[37] K. Lakhotia, P. McMinn, and M. Harman, "Automated Test Data Generation for Coverage: Haven't We Solved This Problem Yet?," in *Testing: Academic and Industrial Conference-Practice and Research Techniques, 2009,* pp. 95–104.

[38] C. Wiederseiner, S. Jolly, V. Garousi, and M. Eskandar, "An open-source tool for automated generation of black-box xunit test code and its industrial evaluation," *Testing–Practice and Research Techniques*, pp. 118–128, 2010.

[39] T. A. Majchrzak and H. Kuchen, "Automated test case generation based on coverage analysis," in *Theoretical Aspects of Software Engineering, 2009. Third IEEE International Symposium on*, pp. 259–266.

[40] Z. Wang, X. Yu, T. Sun, G. Pu, Z. Ding, and J. L. Hu, "Test data generation for derived types in c program," in *Theoretical Aspects of Software Engineering, 2009. Third IEEE International Symposium on*, pp. 155–162.

[41] B. Botella, M. Delahaye, S. Hong-Tuan-Ha, N. Kosmatov, P. Mouy, M. Roger, and N. Williams, "Automating structural testing of C programs: Experience with PathCrawler," in *IEEE ICSE Workshop on Automation of Software Test,* 2009, pp. 70–78.

[42] Carlos Pacheco and Michael D. Ernst, "Eclat: Automatic Generation and Classification of Test Input, "in *Lecture Notes in Computer Science ,*" ECOOP 2005.

[43] Y. Cheon and C. E. Rubio-Medrano, "Random test data generation for Java classes annotated with JML specifications," Technical Report, in *University of Texas at El Paso,UTEP-CS-07-11,* 2007.

[44] T. Figliulo, A. von Mayrhauser, M. Shumway, and R. Karcich, "Experiences with automated system testing and sleuth," in *IEEE Aerospace Applications Conference , 1996,* vol. 4, pp. 335–349.

[45] K. Ghani and J. A. Clark, "Automatic Test Data Generation for Multiple Condition and MCDC Coverage," in *IEEE Fourth International Conference in Software Engineering Advances,* 2009, pp. 152–157.

[46] Z. J. Li, J. Zhu, L.-J. Zhang, and N. M. Mitsumori, "Towards a practical and effective method for Web services test case generation," in *IEEE Workshop on Automation of Software Test,* 2009, pp. 106–114.

[47] S. Mani, V. S. Sinha, S. Sinha, P. Dhoolia, D. Mukherjee, and S. Chakraborty, "Efficient Testing of Service-Oriented Applications Using Semantic Service Stubs," in *IEEE International Conference on*

*Web Services,*2009, pp. 197–204.

[48] A. J. Offutt, "An integrated system for automatically generating test data," in *IEEE First International Conference on System Integration, 1990,* pp. 694–701.

[49] N. Smeets and A. J. H. Simons, "Automated Unit Testing with Randoop, JWalk and muJava versus Manual JUnit Testing.", in *Department of Mathematics and Computer Science in University of AntWerp, 2009.*

[50] A. Von Mayrhauser, J. Walls, and R. Mraz, "Sleuth : a domain based testing tool," in *IEEE International Test conference, 1994,* pp. 840–849.

[51] D. Zamli, K. Zuhairi, D. Mat Isa, and N. Ashidi, "Development Of An Automated Unit Testing Tool For Java Program," Project Report in *Universiti Sains Malaysia,* 2009.

[52] S. Zhang, "Palus: A Hybrid Automated Test Generation Tool for Java," in *IEEE 33rd International Conference on Software Engineering,* 2011, p. 1182.

[53] Michael Ziller, "PeX – Parameterized Unit tests in Visual Studio," in Formal Software Development Seminar,University of Karlsruhe, 2008.

[54] Nikolai Tillmann and Jonathan de Halleux, "Parameterized Unit Testing with Microsoft PeX", Long Tutorial in *Citeseer,* 2010.

[55] Nikolai Tillmann and Jonathan de Halleux, "Pex- White Box test Generation for .Net" in *Tests And Proofs, Lecture Notes in Computer Science, 2008*.

[56] Jonathan Aldrich, "Evaluation of AgitarOne," Analysis of Software Artifacts, Final Project Report, in *Carnegie Mellon University, School of Computer Science,* 2007.

[57] Carlos Pacheco and Michael D. Ernst, "Eclat: Automatic Generation and Classification of Test Inputs," Technical Report MIT-LCS-TR-968, in MIT Computer Science and Artificial Intelligence Lab, 2005.

[58] Carlos Pacheco, Shuvendu Lahiri and Thomas Ball, "Finding errors in .Net with Feedback- directed random testing," in *Proceeding of the International Symposium on Software Testing and Analysis,* 2008.

[59] Sai Zhang, David Saff, Yingyi Bu and Michael D. Ernst "Combined Static and Dynamic Automated Test Generation" University of Washington, Google Inc and University of California, 2011.

[60] Tao Sun, Zheng Wang, Geguang Pu, Xiao Yu, Zongyan Qiu and Bin Gu "Towards Scalable Compositional Test Generation" in 9th International conference on quality, 2009

[61] Rmzelle, "Zotero Documentation", http://www.zotero.org/support/, 10th June, 2012

[62] Dimitris Bertsimas and John Tsitsiklis, "Simulated Annealing", in *Institute of Mathematical* Statistical, 1993.

[63] Tobiah E. Smith and Dorothy, "Using Stimulated Annealing to Synthesize Resource- Bounded Software", in Automated Software Engineering, 1994.

[64] J. Rajesh, V. K Jayaraman and B.D Kulkarni, "Taboo Search Algorithm for Continuous Function Optimization", in *Chemical Engineering Research and Design*, 2000, pp 845-848, ISSN 0263-8762.

[65] Filomena Ferrucci, Carmine Gravino, Rocco Oliveto and Federica Sarro, "Using Tabu Search to Estimate Software Development Efforts", *Software Process and Product Measurement in Lecture*

Notes in Computer Science, 2009

[66] Tom V. Mathew, "Genetic Algorithm", Department of Civil Engineering, Indian Institute of Technology, Bombay, 2005.

[67] Sarfraz Khurshid," Testing an Intentional Naming Scheme Using Genetic Algorithms", *Tools and Algorithms for the Construction and Analysis of System in Lecture Notes in Computer Science, 2001.*

[66] Korel, B. Automated Software Test Data Generation. IEEE Transactions on Software Engineering, Volume 16, No. 8, pp. 870-879, August, 1990.

[67] A. J Offutt, Jin, Z., and Pan, J. The dynamic domain reduction procedure for test data generation. Software Practice and Experience, Volume 29, No. 2, pp. 167-193. 1999.

[68] Tracey, N.J., Clark, J., Mander, K., and McDermid, J. An Automated Framework for Structural Test-Data Generation. In Proceedings 13th IEEE Conference in Automated Software Engineering, Hawaii, October 1998.

[69] Michael, C.C., McGraw, G.E., Schatz, M.A., and Walton, C.C. Genetic algorithms for dynamic test data generation. In Proceedings of the 12th IEEE International Automated Software Engineering Conference (ASE 97), pp. 307-308, Tahoe, NV, 1997.

[70] Marco Dorigo, "Ant Colony Optimization", IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, scholarpedia, 2(3): 1461, 2007

[71] Irman Hermadi, "Genetic Algorithm Based Test Data Generator" in *IEEE Conference of Congress on Evolutionary Computation, Dec 2003*

[72] Bogdan korel, "Automated Software Test Data Generation" in *IEEE Transactions on Software Engineering*, Vol 16, August 1990.

[73] Iyengar .V and Chakrabarty. K "A set of benchmarks for modular testing of SOCs" in *IEEE International Test Conference, proceedings, 2002.*

[74] *Jon Edvardsson, "A Survey on Automatic Test Data Generation" in* Proceeding of 2nd Conference on Computer Science, 1999.

Abbreviations

The Abbreviations used in the research work is shown in the table below:

Abbreviation	Meaning
SLR	Systematic literature review
BPEL	Business process execution language
PICOC	Population, intervention, comparison, outcome and context
MDH	Mälardalens högskola
ACM	Association for Computing Machinery
IEEE	Institute of Electrical and Electronics Engineers
SA	Simulated annealing
TB	Taboo Search
GA	Genetic Algorithms
ACO	Ant colony optimization

Table 15: ***Abbreviations Used in the Report***

APPENDIX A

Appendix A contains the list of links to download the found tools. The table 15 has two fields one for the name of tool and second for the link to download the tool.

Name of tool	Link to Download	Last Accessed Date
Jtest	https://www.parasoft.com/jsp/customers/customer_login.jsp?caller=%2Fjsp%2Ftrial_request.jsp%3FitemId%3D303	2012/06/16
PathCrawler	http://pathcrawler-online.com/doTestYourCode	2012/06/16
DART	http://code.google.com/p/dart/wiki/GettingTheSource?tm=4	2012/06/16
EXE	Not Available	N.A
CUTE	http://cute-test.com/	2012/06/16
SMART	Not Available	
Findsbugs	http://findbugs.sourceforge.net/downloads.html	2012/06/16
KLOVER	Not Available	N.A
SOATest	https://www.parasoft.com/jsp/customers/customer_login.jsp?caller=%2Fjsp%2Ftrial_request.jsp%3FitemId%3D303	2012/06/16
PEX	http://research.microsoft.com/en-us/projects/pex/	2012/06/16
JCrasher	http://ranger.uta.edu/~csallner/jcrasher/	2012/06/16
TestGen4j	http://testertools.com/java-testing-tools/testgen4j-2/	2012/06/16
JUB(Junit test case builder)	http://sourceforge.net/projects/jub/	2012/06/16
Agitar	http://www.agitar.com/	2012/06/16
Eclat	http://groups.csail.mit.edu/pag/eclat/	2012/06/16
Jtst	Not Available	N.A

Austin	http://code.google.com/p/austin-sbst/	2012/06/16
JCute	http://cute-test.com/	2012/06/16
Randoop	http://code.google.com/p/randoop/downloads/list	2012/06/16
JWalk	http://staffwww.dcs.shef.ac.uk/people/A.Simons/jwalk/download.html	2012/06/16
Korat	http://korat.sourceforge.net/downloads.html	2012/06/16
Crest	Not Available	N.A
CAUT	Not Available	N.A
Palus	http://code.google.com/p/tpalus/downloads/list	2012/06/16

Table 16: *Download Tools*