I J C T A, 9(19) 2016, pp. 9211-9219

A Comparative Study of Object-Oriented Software Testing Tools

Rashmi Sharma[*], Anju Saha[*] and A. K. Mahapatra[**]

ABSTRACT

Software testing is one of the most essential steps of software development. Testing is the process of evaluating a system or its modules with the intent to find whether it fulfills the particular requirement or not. Software testing tools facilitate developers and testers to easily automate the procedure of testing in software development. Automated testing tools are competent of executing tests, reporting outcomes and comparing consequences with earlier test runs. Tests conceded out with these tools can be run repetitively, at any point of time. An ample range of software automated testing tools are accessible in the market. Some tools are free while others need a paid license. But it is significant for a user to choose a suitable tool for software testing. This research paper presents a feasibility study based on different parameters of object oriented software testing tools, mainly java testing tools that help developers or users to select the appropriate tool based on their requirements. In this paper, we analyze the concepts and features supported by three open source object oriented testing tools, namely CodePro AnalytiX, JUnit Test Builder (JUB) and Evosuite on the selected target application. The aim of this work is to explore these testing tools to access eccentrically pros and cons of these tools based on certain parameters, so that we can select an appropriate testing tool for the developed software. The main finding is that choice of testing tool depends mainly on application under test (AUT) and learning curve of the tool. If the learning time of the tool is acceptable for the desired goal, then one may select that tool. On the basis of execution of these testing tools on a target application, we have analysed that CodePro AnalytiX is much easier to use.

Keywords: Software Testing, Automated Testing, Test Case Generation, Java, CodePro AnalytiX, JUB, Evosuite, AUT.

I. INTRODUCTION

Software testing is a branch of software development where perseverance is crucial. Software testing is the method of determining software quality by executing the software with appropriate test cases to conclude if the proposed software requirements are being satisfied. The IEEE [1] standard glossary defines testing as the process of analyzing a software item to detect the variances between existing and required situations and to evaluate the features of the software items. Testing is vital because previous work on software testing [2], [3], [4] indicates that software reliability is defined via testing and around fifty percent of the software development cost for software projects is spent on testing [5]. Researchers [6], [7], [8] claim that more effort spent on testing will result in improved quality software. Thus, it is essential to indulge more resources in software testing. Software testing can be done either manually or via automated testing tools. In manual software testing [9], software tester prepares the test cases for various levels of code, executes the test cases and reports the results. Manual software testing is labor intensive, time demanding and very expensive. The effort required will be same each time, as manual test is not reusable. Thus manual testing has limited visibility and needs to be repeated by all stakeholders. Also, some errors may remain uncovered in manual testing. Due to this repetitive and labor intensive behavior of manual software testing, it is necessary to trim down human testing.

[*] USICT, GGSIPU, New Delhi, India, *E-mails: rashmimsit@gmail.com; Anju_kochhar@yahoo.com*

[**] IGDTUW, New Delhi, India

Limitations of manual testing can be overcome by automated testing. In automated testing, tester runs the script on a testing tool that generate the suitable test cases. Tester may or may not be aware of the inside details of the system under test. ISTB [10] defines automated testing as the use of software, to control the implementation of tests, the comparison of actual results to expected results, the setting up of test prerequisites, and other test control and reporting tasks. Automated software testing of software projects automatically verifies their key functionality, tests for regression and can run a large number of tests in a short period of time. On the basis of the license associated with these testing tools, the automated testing tools can be classified as: open source test tools and commercial/paid test tools. Open source test tools are freely available tools as, they don't require any license and can be downloaded from the internet or can be obtained from the vendor without any cost. Also, code of the application is also accessible to user for further enhancements. For commercial testing tools, a license has to be purchased to explore the full functionality of the tool. The cost may vary as per the functionality of the tool. Because of the repetitive nature of software testing, it is necessary to identify appropriate software testing tool for the system under test. In this paper, we discuss three open source object oriented software testing tools namely, CodePro AnalytiX, JUnit Test Case Builder (JUB) and Evosuite and compare them on the basis of few parameters to determine their usability and effectiveness, that helps in the selection of appropriate testing tool. The remaining paper is structured as follows: Section II converses the background work done on software testing tools. Section III describes the methodology used in this research. In this, we provide the details of tool and parameters identified for evaluation. Section IV gives the conclusion part and future scope.

II. RELATED WORK

This section describes the work done by various researchers on software testing tools. J. Meek et al. [11] presented the design and execution of an automated software testing tool. This tool creates test cases automatically for specific three-variable functions. In each test case, an output is generated and compared to a computed expected test result obtained from an input file. This research provides designers with an assurance that the implemented system achieves a very high level of correctness. Meenu and Yogesh Kumar [12] provided a viability study based on different parameters of commercial tools such as the Selenium, SoapUI and open source automation testing tool named HP Unified Functional Testing (UFT), TestComplete (TC). They concluded that user can select a testing tool depending on the type of application to be tested, budget, and the productivity required for that application. Selenium, SoapUI, HP UFT and TC all are extremely fine tools. Each tool has its own advantages and disadvantages. For the application under test, SoapUI is the best tool among the four.

Tarik Sheth and Dr. Santosh Kumar Singh [13] selected three testing tools namely, RFT, Ranorex, and Janova. They concluded that selection of software testing tools depends on the context. It takes time, effort and software testing goal to know which tool is best suitable to use. T.Illes et al. [14] provided a systematic approach to derive an evaluation criteria of software testing tools. They compared three capture and replay tools and concluded that defined criteria can be effectively applicable on evaluation of test tools. Stefen Wagner et al. [15] worked on case study using various projects from industrial environment. They compared three bug finding tools namely, FindBugs, PMD and QJPro and found that bug finding tools can save cost and tools should be improved in terms of false positive ratio and tolerance of different programming styles. Koushik Sen and Gul Agha [16] presented a tool paper on CUTE and jCUTE. Their approach was based on concolic testing,that can execute a program both concretely and symbolically. Catherine Oriat [17] in her paper presented Jartege tool, which generates random unit test cases for Java classes specified in Java Modelling Language (JML). Christoph Csallner and Yannis Smaragdakis [18] discussed JCrasher which is an automatic robustness testing tool for java. JCrasher can be integrated in eclipse IDE. Anthony J.H. Simons [19] in their work discussed a tool named JWalk for systematic unit testing in the perspective of agile methods. Elsa Gunter et al. [20] discussed PET (Path exploration tool) which is an interactive software

testing tool. Elvira Albert et al. [21] presented jPET tool which work as an automatic test case generator for java (TCG).In the next section, we discuss the methodology adopted for the comparison of object oriented software testing tools.

III. METHODOLOGY USED

The research in this paper includes:

1. Selecting a set of tools for evaluation
2. Identifying a set of parameters for evaluation of tools
3. Selecting the target application
4. Testing the target application using each selected tool and gathering the resulting data
5. Results Interpretation

3.1. Selection of Tools

In this research, we are focusing on object oriented software testing tools. Hence, we are using java testing tools. Table 1 provides details of the automated testing tools used in this study. It consists of name and version of the tool used, type of input, user interface of the tool, source in which tool is written, and last attribute specifies the work done by previous researchers.

Table 1
Automated Software Testing Tools

Name	*Version*	*Input*	*Interface*	*Source*	*Research Publications*
Evosuite [36]	Evosuite-1.0.3	Class File	Command Line	Java	[26],[27],[28],[29],[30]
JUnit Test Case Builder (JUB) [37]	0.1.2 (2002)	Source File	Eclipse Plug-in	Java	[31]
CodePro AnalytiX [35]	7.1.0	Source File	Eclipse Plug-in	Java	[22],[23],[24],[25]

3.2. Identification of parameters for the comparison of tools

Identification of parameters provides a way to compare different tools effectively. On the basis of this comparison, user can select an appropriate tool as per his requirements. Here we are classifying our parameters in to the following metrics:

1. Tool Feature Metric
2. Tool Usability Metric
3. Tool Debugging Metric
4. Tool Requirement Metric
5. Tool Performance Metric

These metrics are influenced from the work done by N. Bordelon [32] on comparison of automated software testing tools. Table 2 provides the details of identified metrics and parameters measured under them.

3.3. Target Application

In this study, Java open source software “Apache Lucene” is used for the comparison of testing tools. Apache Lucene is [33] a high-performance, full-featured text hunt engine library. Many versions of Lucene are available, we have used Lucene 4.10.2 core API. Its specifications are given in Table 3:

Table 2
Parameters Used for Comparison of Testing Tools

S.No	*Type of Metric*	*Description*	*Name of Parameter*	*Description*
1.	**Tool Feature Metric**	This metric provides all the basic functionalities supported by the tool. These functionalities are present out-of-box in the tool and require no intervention from the user or developer. This metric basically captures the functional requirements of the software.	**Installation Required/Cloud Based**	This parameter describes whether installation is required for the tool or it is available as a cloud based tool.
			Programming Knowledge	This parameter determines whether knowledge of programming/Scripting language is required or not.
			List of Features	This parameter lists all The Features/Functionalities supported by the tool.
2.	**Tool Usability Metric**	This metric determines, with how much ease one can use the functionalities of the tool. This metric determines the non-functional requirements of the software and hence determines the learning curve for the tool.	**Ease of Installation**	This parameter determines, how easy or difficult it is to install the tool.
			User Friendly Interface	This parameter determines, how easy or difficult it is to use a particular tool.
			Availability of Tutorial	This parameter determines that for a particular tool "Instruction manual" exits or not.
			Understanding of error messages	This parameter determines, how easy or difficult it is to understand the error message generated by the tool.
3.	**Tool Debugging Metric**	This metric determines the ease with which we can debug our code.	**Documentation of error message Logs**	This parameter determines, how well documented are the error logs. With how much ease user can debug the code from error logs.
			Generation of Test Cases	This parameter determines, how easily test cases can be generated with the tool.
4.	**Tool Requirement Metric**	This metric lists all the mandatory requirements needed for the tool to install and run the software.	**Programming Language**	This parameter determines the programming language required to run the tool effectively.
			System Requirements	This parameter determines the operating system, hardware and software required for the tool to run.
			Testing Environment	This parameter determines whether the testing environment is based on Command line interface or GUI.

Table 3
Apache Lucene Specifications

Project Used	*System Requirements*	*Number of Packages*	*Number of Classes*	*Lines of Code*	*Number of Fields*	*Number of Methods*	*Number of Lines*
Apache Lucene[33]	Java Version 6 or higher	33	727	98,186	5029	7991	168,699

3.4. Testing Apache Lucene

All the three software testing tools have been used, evaluated and compared on Apache Lucene. In this section, we discuss the testing of apache Lucene on the three selected testing tools.

3.4.1. Evosuite

Evosuite tool develops unit tests for java software. It is a Search based testing tool (SBST) which uses genetic algorithm to generate test suites. It develops entire test suites targeting all coverage goals at the same time. Figure 1 describes the evosuite process of test suite generation.

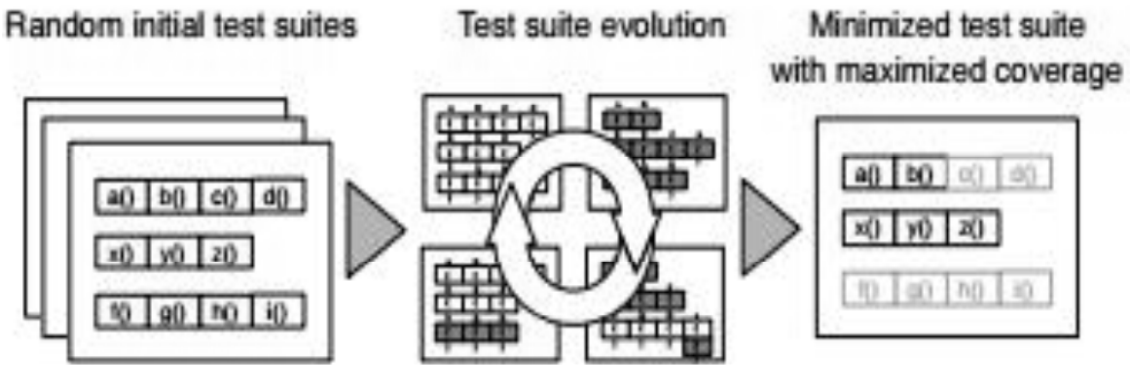

Figure 1: Evosuite Process [33]

We run evosuite from command line. The following command is used to generate test cases from the compiled apache lucene source:

Java –jar evosuite-1.0.3.jar –projectCP= "C:\Users\skumar3\Desktop\phd\evosuite demo\lucene" –target= "C:\Users\skumar3\Desktop\phd\evosuite demo\lucene"

It will create two files: evosuite-reports and evosuite-tests.

1) Evosuite-reports contains statistics.csv file, which has target class, criterion used, coverage, total goals and covered goals as its attributes. The specifications of these attributes are as follows: a) Target class specifies the class under test. b) Coverage criteria are used to guide test generation. A coverage criterion signifies a finite set of coverage goals, and a common method is to target single such goal at a time, creating test inputs either symbolically or with a search-based approach. c) Coverage is a measure that describes the degree to which source code of a program is tested by a particular test suite.
2) Evosuite-tests, produce two files for every class under test. For example: For IndexFiles.class ,it creates: evosuite-tests\org\apache\lucene\demo\ IndexFiles_ESTest.java and evosuite-tests\org\apache\lucene\demo\ IndexFiles_ESTest_scaffolding.java

The scaffolding confirms that tests are always executed in the same stable state, so they should fail only if they reveal a bug. Tests are in the IndexFiles_ESTest.java file. Figure 2 shows the demonstration of evosuite on apache lucene.

3.4.2. JUnit Test Case Builder (JUB)

Junit Test case builder is installed on Eclipse-Helios and Junit test cases are generated for each class file. JUB's framework is built on Builder pattern (GoF) with white box test generation support. This tool generates test cases having 0 for integers and null for other input variables. Figure 3 shows the generation of test cases via JUB. After this step, package name prefix with "unittest" is created which has the test files. For example, in this case a package named unittest.org.apache.lucene.demo is created which thus contains IndexFilesTest.Java file.

3.4.3. CodePro AnalytiX

CodePro AnalytiX plugin is installed on eclipse-helios and test cases are generated. The CodePro JUnit Test Case generation ability helps us to automate the design of complete JUnit test cases. It does this through a blend of both static code analysis and by dynamically implementing the code to be tested in order to detect the behavior of code. Test cases will be generated as a separate project. Figure 4 shows the way to generate test cases using CodePro AnalytiX.

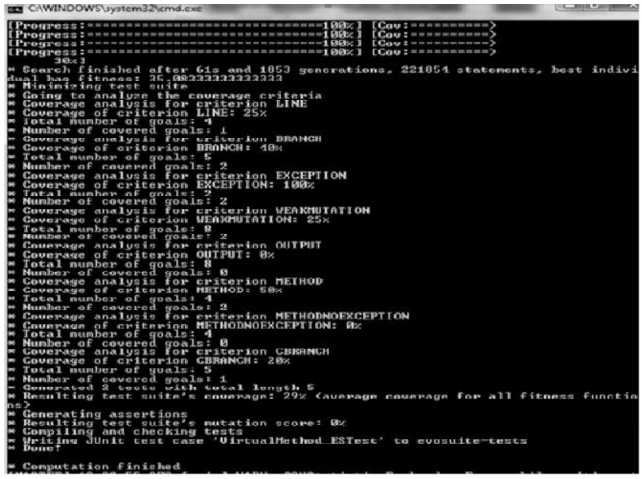

Figure 2: Generation of Test Cases via Evosuite

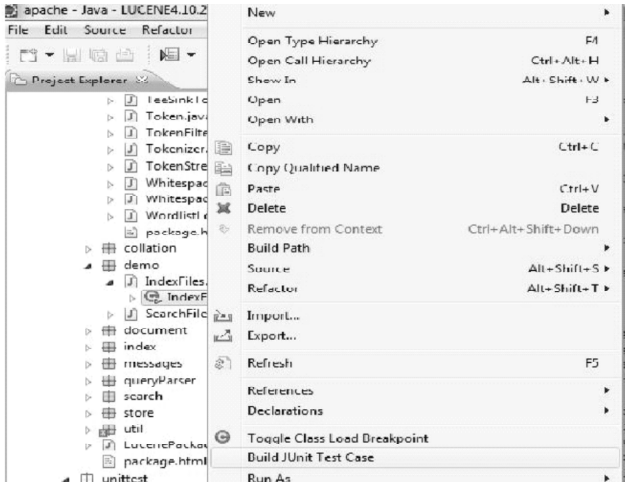

Figure 3: Generation of Test Cases via JUB

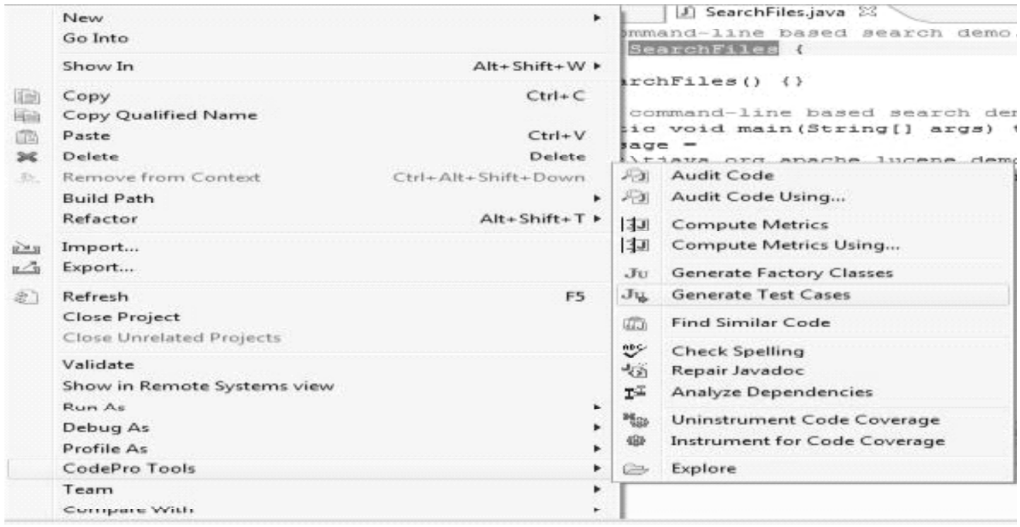

Figure 4: Generation of Test Cases via CodePro AnalytiX

3.5. Results Interpretation

Now, we compare these tools on the basis of above mentioned metrics. Table 4 summaries the comparison:

Table 4
Comparison of Software Testing Tools

1) Tool Feature Metric

Parameter Name	Evosuite	JUB	CodePro AnalytiX
Installation Required/Cloud Based	Yes	Yes	Yes
Programming Knowledge	Knowledge of command-line commands is required	Yes	No
List of Features	Generation of JUnit 4 tests for the selected classes, Optimization of different coverage criteria.Tests are minimized.Generation of JUnit asserts to capture the current behavior of the tested classes.	JUnit test case generator, Independent of externally enforced environment.	Defect Detection, Repair and Reporting,Automated JUnit Test Creation, JUnit Test Editor, Code Coverage Exploration, Static Code Investigation.

contd. table 4

2) Tool Usability Metric

Parameter Name	Evosuite	JUB	CodePro AnalytiX
Ease of Installation	For Command-line: Require Knowledge of command line syntaxes to execute jar file. In eclipse, it is also required to M2 eclipse plugin installed.	The jub.zip file accessible for download has jub.jar and plugin.xml files. These files should be placed in eclipse plugin folder.	Can be installed in eclipse using the url: http://dl.google.com/eclipse/inst/codepro/latest/3.6[35]
User Friendly Interface	Yes (Eclipse)	Yes	Yes
Availability of Tutorial	No basic tutorial is available.	No tutorial is available. Website is also very raw. http://jub.sourceforge.net/ [37]	Evaluation guide is available, which is very helpful and easy to use.
Understanding of Error Message	Error messages are not very clear and not properly documented.	Error messages are properly documented.	Error messages are properly documented.

3) Tool Debugging Metric

Parameter Name	Evosuite	JUB	CodePro AnalytiX
Documentation of error message Logs	Not well documented.	Error logs are well documented but some errors may arise randomly and thus need eclipse restart.	Logs are very well documented, but exact root cause of some errors occurred during testing is very tedious to find.
Generation of Test Cases	Following command is used to generate test case:- Java –jar evosuite-1.0.3.jar -projectCP= "C:\Users\skumar3\Desktop\phd\evosuite demo\lucene" –target= "C:\Users\skumar3\Desktop\phd\evosuite demo\lucene"	Select a particular class for which test case need to be generated.	Select a particular package for which test case need to be generated.

4) Tool Requirement Metric

Parameter Name	Evosuite	JUB	CodePro AnalytiX
Programming Language	Java	Java	Java
System Requirements	Support Eclipse (Plugin) and require JDK environment, if running via command line	Supports Eclipse and VisualAge for java	CodePro seamlessly integrates into any Eclipse-based JDK environment.
Testing Environment	Command line	Windows GUI	Windows GUI

IV. CONCLUSION AND FUTURE SCOPE

As software testing tools are very important part of software testing process and it is very critical to select the appropriate tool for testing. It requires an ample amount of time and effort to understand any particular tool. Also, the selection is based on Application under test (AUT). The ideal testing tool should be one, with good user interface, easy to install and for which sufficient number of tutorials are available. In this study, we compare Evosuite, JUB and CodePro AnalytiX on a set of parameters. A well framed evaluation guide is available for CodePro AnalytiX, but for JUB or Evosuite only some web links are there for help. Thus, we found the learning curve of CodePro AnalytiX much smaller than Evosuite and JUB. Moreover, CodePro AnalytiX provides additional features like computation of metrics, analysis of dependencies among packages, auditing of code. These additional functionalities make it more usable and effective than its counterparts. Also, CodePro AnalytiX displays all the issues in currently opened editor itself, which is much more appropriate than selecting packages or projects and then trigger a scan on them. Thus, in this study, we found that CodePro AnalytiX is much simpler and more powerful than JUB and Evosuite. Our future research will focus on providing a survey of both commercial and open source object oriented software testing tools with more refined criteria. We plan to extend this research on atleast ten open source object oriented projects, so that we may get a more general result.

REFERENCES

[1] IEEE Press, IEEE Standard glossary of software engineering technology, ANSI/IEEE Std. 610.12., 1990.

[2] R. Brown and M. Lipow, "Testing for software reliability" ,Proc. ACM Int. Conf. on reliable software, pp. 518-527, New York, 1975.

[3] H. Pham., "Software reliability and testing", IEE Computer society press, 1995.

[4] A.Bertolino, "Software testing research:achievements,challenges,dreams", Future of software engineering(FOSE), IEEE Computer society, 2007.

[5] Beizer, "Software testing techniques (second edition)", Van Nostrand Reinhold Co., 1990.

[6] Joseph P.Cavano and James A. McCall, "A framework for the measurement of software quality", Proc. ACM software quality assurance workshop on functional and performance issues., New York, 1978.

[7] D. Hamlet, "An essay on software testing for quality assuance", Annals of software engineering, vol.4, pp.1-9, Springer, 1997.

[8] J. R. Horgan,S.London and M. R. Lyu, "Achieving software quality with testing coverage measures", Computer, IEEE Computer society, vol. 27, pp. 60-69, 1994.

[9] P. Ammann and J.Offutt, "Introduction to software testing", Cambridge university press, 2008.

[10] International software testing qualifications board, "Standard glossary of terms used in software testing, version 3.01".

[11] J. Meek,N. Debnath, I. Lee and H. Lee, "Algorithmic design and implementation of an automated testing tool", IEEE Int. Conf. on Information technology, pp. 54-59, 2011.

[12] Meenu and Y. Kumar, "Comparative analysis of automated functional testing tools", Int. Journal of emerging technologiesand innovative research, vol. 2, pp. 42-48, 2015.

[13] T. Sheth and Dr. S. K. Singh, "Software test automation-approach on evaluating test automation tools", Int.Journal of scientific and research publications, Vol. 5, 2015.

[14] T. Illes, A. Herrmann, B. Paech and J. Ruckert, "Criteria for software testing tool evaluation-a task oriented view", Int. software quality institute,Proc. of the 3rd world congress for software quality, vol. 2, pp. 213-222, 2005.

[15] S.Wagner, J. Jurjens, C. Koller and P. Trishberger, "Comparing bug finding tools with reviews and tests", Testing of communicating systems, springer, LNCS, vol. 3502, pp. 40-55, 2005.

[16] K. Sen and G. Agha, "CUTE and jCUTE: Concolic unit testing and explicit path model-checking tools (Tool paper)", CAV, LNCS, vol. 4144, pp. 419-423, 2006.

[17] C. Oriat, "Jartege: A tool for random generation of unit tests for java classes", QoSA-SQQUA, LNCS, vol. 3712, pp. 242-256, 2005.

[18] C. Csallner and Y. Smaagdakis, "JCrasher:An automatic robustness tester for java", software –practice and experience, vol.34, pp. 1025-1050, 2004.

[19] A. J. H. Simons, "JWalk: A tool for lazy,systematic testing of java classes by design introspection and user interaction", Automated software engineering,vol. 14, pp. 369-418, springer, 2007.

[20] E. Gunter, R. Kurshan and D. Peled, "Per" An interactive software testing tool", CAV, LNCS, vol. 1855, PP. 552-556, 2000.

[21] E. Albert et al., "jPET:An automatic test case generator for java", Proc. 18th working conference on reverse engineering,IEEE Computer society, pp. 441-442, 2011.

[22] R. Terra, L.F. Miranda, M.T. Valente and R.S. Bigonha, "Qualitas.class corpus:A compiled version of the qualitas corpus", Sigsoft software engineering notes 38, pp. 1-4, 2013.

[23] M. Sensalire, P. Ogao and A.Telea, "Classifying desirable features of software visualization tools for corrective maintenance", Proc. ACM symposium on software visualization, New York, 2008.

[24] R.Plösch, A. Mayr, G. Pomberger and M. Saft, "An approach for a method and a tool supporting the evaluation of the quality of static code analysis tools", SQMB, 2009.

[25] A. K. Tripathi and A.Gupta, "A controlled experiment to evaluate the effectiveness and the efficiency of four static program analysis tools for java programs", Proc. 18th Int. Conf. on evaluation and assessment in software engineering, ACM, New York, 2014.

[26] G. Fraser and A. Arcuri, "Evosuite:Automatic test suite generation for object-oriented software", Proc. 19th ACM Sigsoft symposium and 13th european conference on foundations of software engineering, ACM, New York, 2011.

[27] G. Fraser and A. Arcuri, "Evosuite at the SBST 2016 tool competition", Proc. 9th Int. workshop on search based software testing, ACM, pp. 33-36, New York, 2016.

[28] J.M. Rojas, G. Fraser and A. Arcuri, "Automated unit test generation during software development:A controlled experiment and think-aloud observations", Proc. Int. Symposium on software testing and analysis, pp. 338-349, 2015.

[29] G. Fraser, A. Arcuri and P. McMinn, "A memetic algorithm for whole test suite generation", Journal of systems and software, vol. 103, pp. 311-327, 2015.

[30] Y. Pavlov and G. Fraser, "Semi-automatic search based test generation", IEEE Fifth Int. conference on software testing,verification and validation.

[31] S.Wang, "Comparison of unit level automated test generation tools",software testing,verification and validation workshop, 2009.

[32] N. Bordelon, D. Reinicke, B. Patterson and L. Noble, "A comparison of automated software testing tools",annals of the master of science in computer science and information systems at UNC Wilmington.

[33] G. Fraser and A. Arcuri, "Evolutionary generation of whole test suites, Proc. 11th Int. conference on quality software,pp.31-40,2011.

[34] *http://lucene.apache.org*

[35] *https://marketplace.eclipse.org/content/codeproanalytix*

[36] *http://evosuite.org*

[37] *http://jub.sourceforge.net*