Comparison of Automated Unit Testing tools

Aasminpreet Singh Kainth

Department of Computer Science
Middlesex College, University of Western Ontario
London, Ontario, Canada.

akainth4@uwo.ca

Abstract - Unit testing is an important and effective technique to improve the code in terms of quality and usage. The main idea behind the unit testing technique is that each snippet of code should be tested with its own tests. But, generating these tests is a very time-consuming job. So, in order to reduce this time-consuming job automated unit test generation tools are used. It is observed, as unit testing is gaining more attention by the developers similarly these automated unit test tools are coming in more demand. But then the problem comes with choosing of an appropriate tool considering many factors such as availability of the tools, Language supported, etc. The main intent of this paper is to categorize and tabulate the tools for solving the problem of developers to choose the specific tool. This paper could also be beneficial for the researchers as they can use the information given in the paper to build new tools and to study and evaluate the effectiveness of the tools mentioned.

I. Introduction

In Software Testing, Unit testing is regarded as an important and effective testing technique to improve the code efficiency and effectiveness in terms of quality and usage. The main idea behind the unit testing technique is that each snippet of code should be tested with its own tests. The purpose of unit testing is to test each snippet of code whether it is giving desired results or not. The term unit refers to the smallest testable part in the code or software. A unit may have multiple inputs but a single output.

A. Problem Description

In software development, unit testing gives the developer the opportunity to test each unit of the software where it is observed that many problems are found just because of the unit testing. A successful software project has complete unit test suite that can run together with each unit test running at the same time. [4]

However, some developers still do not do much unit testing where the reason could be

1. developers do not think the effort and time will be worthwhile.
2. unit tests must be maintained, and maintenance for unit tests is often not budgeted.
3. developers may not know how to design and implement high quality unit tests. [4]

B. Importance

Instead of manually write the unit test cases, Automated unit test generation tools plays an important role in the software industry as it can help solve the above mentioned problems:

1. These tools can help reduce the time and effort needed to implement and design unit tests
2. As if the program is changed, automated test generation tools can be used easily to maintain the code.
3. It's a win-win situation for the developers to use these tools as these can encapsulate the knowledge to design and implement high quality tests. So, in this case developers do not need to know as much.

But an important question that each developer should answer is that "which tool should be used in a specific environment?" [4]

C. Motivation

The major motivation in doing this research is to solve the problem of choosing the appropriate unit test tool as mentioned above it can lower the time involved in actual testing by the developers and clearly lower the cost of performing unit testing.

The problem is how to choose the most appropriate tool that will suit developer requirements consisting of [14]

1. cost involved
2. effort needed
3. level of automation provided
4. language support

The rest of the paper is organized as follows. Section 2 provides the background information of the papers read. Section 3 outlines the analysis for my work. The discussion to the research is presented in Section 4, while Section 5 aims to conclude. Section 6 gives the references to research papers discussed.

II. Background

Testing is always considered to be laborious, expensive and error prone. But the solution to this problem is Automation, and the pivot role in test automation is played by generating test data automatically. In [12], Yoonsik Cheon and Carlos E. Rubio-Medrano gave the proposal to a new approach for generation test data randomly. These test data generation was done in Java classes with JML specifications. In other words, they have explained the hidden state problem of objects on generating test by addressing the problem of randomly chosen sequence of method calls. Their approach checked the validity of each method call as when the call is selected. Their implementation applies new test data generation strategy to a tool known as JET for Java where JET, fully automates the unit testing.

The results of their approach [12] used is positive as it can increase the feasible sequence construction possibility which further can help reduce the redundancy. The author outlies some of the cons of JET in the paper, though the cons mentioned are only considered while the current implementation. JET doesn't support object sharing. The contributions of [12] include checking equivalence of objects and its specs, pooling of objects and making use of runtime checker.

Their work also discussed some other automated test generation tools such as JCrasher which is an automated random testing tool used to test the robustness. The authors of [12] also considered the commercial tool from Parasoft named Jtest that supports automatic white box testing.

S Wang and J Offutt in [4] compared some well known automated unit test tools that are publicly available, JUB, TestGen4j and JCrasher. These tools were evaluated on the basis of mutation scores. In their paper, they described Junit test case Builder as Junit test case generator framework; TestGen4j which focuses boundary value analysis. Other description of tools used are given in the tables in the Analysis Section. On comparisons they found JCrasher generated many tests which are of no use

in the testing which seems to put extra effort on the developers to evaluate the results from those tests. They concluded their paper by writing that tools used for comparisons were lame at finding the bugs.

In the article [11], authors have presented the tool for c and cpp softwares for structural test generation. The test generation method used in PathCrawler tool has shown the positive effects in test case generation covering different paths. PathCrawler maintains the state of each symbolic execution of the program. This tool is able to generate test cases over 1,000,000 paths that can have 100s of the control points of a function under test. This paper also compared the Pathcrawler with other tools such as CUTE, EXE and PEX. The test generation method given by PathCrawler shows positive output. An important method for the evaluation of an automatic test generation tool is done by the performance of tools.

Euclide, a constraint- based testing tool is introduced in paper [2]. This tool takes the C code for test data generation. The tool introduced has many applications such as partial program proving, structural test data and counter example generation. The tool has advantages. Arnaud Gotlieb done several experiments on the tool to analyse the tool, initially they evaluated structural test data generation for all decision criteria. Arnaud also discussed various other tools such as PathCrawler, Dart, CUTE, and regarded these tools to be more oriented on concolic testing and path selection. Whereas the Euclide has goal oriented approach. The improvements in the tool is possible and requires extra effort to research so that to increase efficiency and effectiveness of the Euclide.

The paper [10] shows the need of automation test tools in the Blackbox unit testing as testing in this type is very expensive. Blackbox unit testing tests the functionality of the code without knowing the actual code so the number of possible test cases is huge to consider and needs the automated tools to solve this problem. The authors of [10] comes up with the Auto Blackbox Unit testing tools which is considered to be efficient and effective in automated generation of test cases. The authors have analysed their tool on the results of testing the mutated code and compared the accuracy of their tool with other available tools in the market solving the same purpose such as JUB, TestGen4j and JCrasher.

For future work, authors decided to test AutoBBUT tool for comparisons with some more tools. The authors have also planned to compare the effort and time spent and efficiency to the manual coding of unit test if AutoBBUT is used in large-

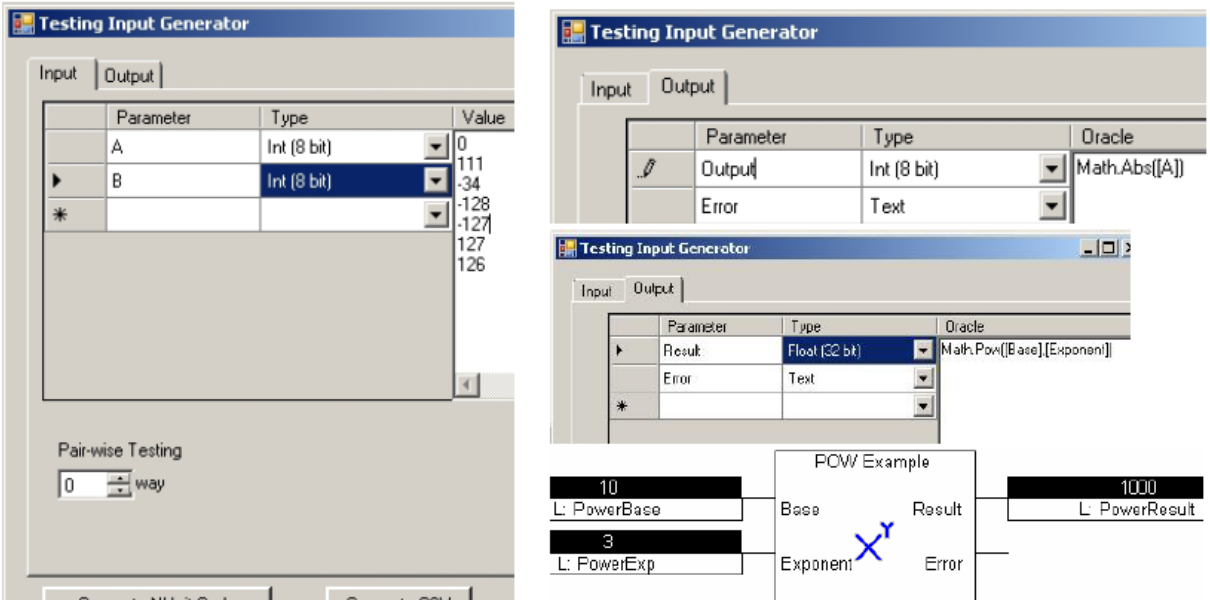

Figure 1 Snapshots from the AutoBBUT's GUI

scale projects. In [10], authors plan to apply mutation testing for the further analysis and assessmentof fault detection effectiveness. They also plan to compare effectiveness with other tools such as JUB, TestGen4j and JCrasher.

N Smeets and Et al. conducted a study on three unit testing tools which are Randoop, Jwalk and uJava. The tools used for comparisons in [13] has different testing methods. In their study, they have summed up different pros and cons of these tools. JWalk tool constructs tests with combinations of code analysis. JWalk work directly with the compiled Java class files. Whereas, ujava require ample amount of time for testing which seems to be unreasonable. In authors opinion, hand-written test cases were better than the test cases by the automated test tools. The tools used in [1] are CUTE, Dart. PathCrawler and EXE.

The authors in [7], described a prototype tool called SimC which can generate test data for unit testing automatically for C softwares. The tool uses the pointer operations. Their tool shows effectiveness in handling real-world software testing scenario. SimC can generate test data for software made up of complex pointer operations. They have drawn the comparisons with other tools such as DART, and CUTE. They also plan to expand their tool further so that more code could be handled.

III. Analysis

This section evaluates the data collected from the literature review process. The list of papers collected has aimed to find the unit-testing tools with automation support available in the market. The table 1 below contains the list of tools available in the market with the respective cited paper(s). The

S No	Tool name	Citation(s)
1	Agitar	[4]
2	Austin	[5], [9]
3	CREST (Successor of CUTE)	[5]
4	CUTE	[5], [9], [1], [3], [7], [11]
5	DART	[6], [1], [5], [7], [11]
6	Eclat	[4]
7	EXE	[1], [11]
8	Findbugs	[16]
9	Jcrasher	[4], [10], [12]
10	Jcute	[5], [9]
11	Jtest	[4], [12]
12	JUB (Junit test case builder)	[4]
13	Jwalk	[13]
14	PathCrawler	[11], [1], [3]
15	PEX	[3], [5], [10], [11]
16	Randoop	[13]
17	SMART (Extension of DART)	[1]
18	TestGen4j	[4], [10]

Table 1 List of Unit Testing Tools with Automation Support

table 1 is divided into three columns S. No., tool name, and citation.

The citation denotes that either the paper presents a detailed comparison of the tool or the tool is introduced in the paper. These citations 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, the total number of tools found is 18. There are some tools mentioned in the table that do not have any primary citation(s), those are normally commercial tools and are not freely available.

Table 1 shows, the list of found results in alphabetically order, but this table does not give any more details about the tools. It can be drawn that among non-commercial tools CUTE, DART, and PEX are the three tools which found to have maximum citations. JCrasher is also popular but it is now improved to new version called as Randoop.

PROTOTYPE TOOLS:

During the search process for automation supported unit testing tools, I found some prototype tools developed by some educational group or researchers. In total, I found 4 prototype tools. The mentioned tools in the table 2, 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.

In the table 2, three out of four tools have the only one citation. Among all the four tools, Godzilla is the only tool with two citations.

S No	Tool name	Citation(s)
1	AutoGen	[8]
2	Godzilla	[2], [4]
3	SimC	[7]
4	Symclat	[4]

Table 2 List of Prototype Tools

IV. Discussion

CATEGORIZATION OF TOOLS

Once all the automation supported unit-testing tools are collected, as shown in the table 1. I categorized them and put them into groups. So, before starting the categorization of tools, it is explicit to formulate well-defined categorizing criteria. The categorizing criteria becomes more important to define on collected list of tools because the list contains all the tools regardless saying anything about their domain, language support etc.

A. Based on Domain

This section categorizes the found tools based on the domain it belongs. The list collected in table 1 contains tools irrespective of any domain such as server or desktop. Precisely, domain defines as the type of application that is possible to test with the tool. The most common domains identified are Desktop and server based. From table 3, most of the tools support only desktop-based applications.

There are few points that are decided to include in the categorizing criteria which are discussed below.

Tool Name	Language supported	Testing Technique used	Domain belongs	Availability
Agitar	Java	Observation driven testing	Desktop	Commercial
Austin	C	Search based technique	Desktop	Open source
CUTE	C/C++	Concolic testing	Desktop	Free
Crest	C	Systematic dynamic test generation	Desktop	Open source
DART	C	Concolic testing	Desktop	Open Source
Eclat	Java	Classification technique	Desktop	Academic
EXE	C	Concolic testing	Desktop	Open Source
Findsbugs	Java	Static analysis	Desktop	Open source
Jtest	Java	White box testing	Desktop and Server edition	Commercial from Parasoft
JCrasher	Java	Robustness testing or random testing	Desktop	Open source
JUB (Junit test case builder)	Java	Builder pattern	Desktop	Open source
JWalk	Java	Specification based testing	Desktop	Academic
JCute	Java	Search based and Concolic testing	Desktop	Open source
PathCrawler	C	Concolic testing	Desktop	Online Server
PEX	C#	Dynamically Symbolic exec	Desktop	License from Microsoft
Randoop	Java	Feedback- directed random testing	Desktop	Open source
SMART	C	Concolic testing		Open Source
TestGen4j	Java	Boundary level testing	Desktop	Not Available

Table 3 Categorization of Tools

B. Programming Language Support

It helps to categorize the tools based on the programming language supported. Each tool has the support for at least one specific programming language, but one tool has the support for multiple languages. The most common programming languages concluded form the table 3 are Java and C. In the table 3, only one tool i.e. 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++.

C. Testing Technique

There are many testing techniques exist such as white-box, black box, path testing. All these collected tools in table 1 are directly or indirectly based on any of them. This part will help us to categorize the tools based on the testing techniques used by that specific tool. It is interesting to see in table 3, that tools are based on many different types 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.

D. 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 3, there are 2-3 commercial tools available such as Agitar, Jtest, 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.

Types of errors that each tool can find:

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. The reason I could not find

Tool Name	Type of Error(s)
CUTE	Error in mathematical calculation even inside nested statements; Unnecessary calls of same method; Results comparison with expected values; Null point exception and out of bound in array; Unused code Specialized rules (J2EE); Formatting in the code
DART	Program crashes; Assertion violation; Non termination; Memory allocation detect with collaboration with other run time checking tool
EXE	Constraint Solving; Independent constraint optimization; Bit-victor arithmetic
ECLAT	Out of bound Exception; Pre/post condition violation; Illegal inputs
Jcrasher	Robustness failure; Undeclared run time exception; Pre/Post condition violation Out of memory error; Analysis of exception for error
Jcute	Deadlocks; Uncaught Exceptions; Infinite loop
Jwalk	Unexpected interaction among methods such as dead ends on the fly, broken pre condition; Interleaved constructor
Path Crawler	Infinite loop detection; Pre/Post condition violation; Handling floating point numbers and arithmetic operations; Aliasing and pointer arithmetic
PEX	Argument exceptions; Out of bound in array; Arithmetic specification; Missing pre/Post conditions; Buffer overflow or Resource leak; Syntactic programming error
Randoop	Run time assertion violation; Runtime access violation; Missing messages in resource file and state of resource file; Memory management errors; Concurrency error such that related to test input

Table 4: Types of errors that each tool can find

the type of errors for some tools was the lack of research papers read. 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 research papers.

Most common errors as seen in Table 4 that can be 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. In summary, it can be concluded that the tools are covering wide domain of errors.

V. Conclusion

The 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. The list of tools and categorization table could be very useful for someone who wants to choose automated unit testing tool. The research also provides the list of prototype tools that are either underdeveloped or presents the idea to build tool. As this research work categorized the automated test generation tools so I feel it would be of really great help for software developers in saving their time choosing the nearly perfect tool and increasing the effectiveness for their testing tasks. Additionally, researchers can combine two or more tools to form new tools.

VI. References

[1] 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.

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] I. Singh, “A Study of Automation Tools for Unit Testing” M. U., S. I. D. E., June 2012

[15] Unit Testing [Accessed on 19.12.2019] http://softwaretestingfundamentals.com/unit-testing/ [Dec 2019]