A Survey Report on the Tools of Mutation Testing available in Market

Engr. Afzal Ahmed Masters Student at BUI
afzal.ahmed@email.com

Abstract: Prevailing code coverage techniques in software testing such as condition coverage, branch coverage are thoroughness indicators, rather than the test suites capabilities to detect the fault. Mutation testing can be prospected as a fault based testing technique which can extent the efficiency of test cases for localizing the faults. Producing and running large number of mutants for the test cases is laborious and takes time as well. Therefore, mutation testing is not used as widely as it should in industry. Hence, an automated, fast and reliable tool for the same is required for the better performance is required. Various Mutation testing tools exists for the software industry. In this paper, various available tools which offer object-oriented operators are studied and based on the study; a comparison is made between these Mutation testing tools. An inference is made that all the available tools are language dependent or need to be configured differently for different languages to generate and run test cases. Comparative study reveals that most of available Mutation testing tools are in Java language and possess many important features while less number of tools is available for other languages like C, C++ and C#.

Keywords: Mutation Testing, Mutation Testing Tools, Comparison.

INTRODUCTION

To assess the quality of test cases, one of the most proficient techniques is Mutation Testing. It is a fault based testing technique which measures the efficiency of test cases to localize the faults, on the basis of fault seeding [7]. The idea was first suggested by Richard Lipton in 1971 [5]. In mutation testing the tester makes several copies of the system under test injecting some syntactic changes called faults. The defective copies of System under Test (SUT) are termed as Mutants. Afterwards, these test cases are run against SUT and the mutants to examine how many of the faulty versions of system under test are found by the tests.

A High quality test case is that which can find more number of faulty versions of system under test. The score of Mutation associated with a test suite 'TS' and mutants 'M' is simply given by equation (1); Mutation score = (Number of killed M's / Total M's-Equivalent mutants)*100 (1)

In the paper, various Mutation testing tools are analyzed. All the studied tools are language dependent or they need to be configured differently for different languages. This paper shows the empirical study of Mutation Testing tools available in the market.

All the tools were compared on the various characteristic features. The differentiating features of Mutation testing tools for Object-Oriented languages, includes; 1) Used for languages, 2) Order of Mutation, 3) Manually Selection of Operators. Moreover the distinguishing features among Java supported Mutation testing tools incorporates Object-Oriented operators, Mutants generation level, traditional mutation operators, JUnit support, Type of System, and Type of Mutation testing they perform.

The paper is arranged as follows: Section 2 contains information of work related to Mutation testing. Section 3 provides empirical study about the existing tools of Mutation testing for different languages. Section 4 presents the comparison of the studied tools on the basis

of their different characteristics and in the form of pie charts. The paper is concluded with Section 5.

RELATED WORK

The original idea of Mutation testing was introduced by Richard Lipton in 1971. Lipton, DeMillo, and Sayward developed and published for the first time [7]. Mutation testing tool was first implemented by Timothy Budd as a part of his PhD work (titled as Mutation Analysis) from Yale University in 1980 [5]. There is rich research history on mutation testing [6] and focuses on four different types of activities [4]; 1) To define operators of mutation, 2) testing the operators, 3) development of tools, 4) Decreasing the mutation analysis cost.

Mutation operators are defined for different languages and different testing types in first research activity [6-10]. The second activity is actually experimentation with mutations [5, 11, and 12].. Mutation testing is considered to be way more powerful than branch and statement coverage [9] and finds fault more effectively than the data flow [14, 15]. Offutt et al. [5, 16], Wong and Mathur [10] assessed the selective mutation idea which recognizes those mutation operators that deliver almost the same coverage of testing as a non-selective. By this approach computational cost can be decreased as we have to deal with less mutants. Andrews et al. 231 (Andrews J.H.)suggested that we can get good indication about the capability of detection of faults for a test suite using Mutants. Andrews et al. compared and assessed the criteria of testing coverage by using mutation analysis. Development of Mutation tools is third kind of research activity. MILU [11], Insure++ [12], Proteum [13], Nester [14] were developed for C, C++ and C# respectively. Last activity of research activities investigates the ways to reduce mutation analysis cost. The foremost cost of mutation analysis is because of this fourth activity involving, computational expense in introducing and running wide numbers of mutant programs. Several techniques are mentioned in paper.

Mutation testing is strong technique but consumes a lot of time which makes it a little impractical to use without some kind of fast, reliable and most importantly automated tool that can make a system generate mutants, can run the mutants against a set of test cases, and can report the mutation score of those test cases. Most approaches use one of the given techniques to minimize the cost of mutation analysis [6]; 1) "do fewer": find ways to generate and run less mutant without any loss of information , 2) "do smarter": find ways to distribute mutation analysis over number of machines 3) "do faster": Seek ways to generate and run each of the mutant program faster.

Since all the examined tools are language dependent, they sometimes need to be configured differently for various languages on different platforms. This makes them enable to produce and run the test cases on SUT. The empirical study reveals that maximum of the available tools supports Java language whereas minimum number of the tools supports other languages. The paper shows the comparative study of Java based Mutation Testing tools and tools that can be used for other languages also. An inference is made that Java have the highest number of Mutation Testing tools. This comparison can be differentiated from the tables provided in the subsequent sections.

DETAILED STUDY OF THE AVAILABLE TOOLS

The important features Mutation testing tools have, are that the tool support mutation operators and languages. Object-Oriented (OO) language can distinguish two types of mutation operators suggested by Ma et al [22]. Firstly, the traditional operators [4] reformed from procedural languages. Secondly, object-oriented or class mutation operators [22] developed to handle particular features of OOP.

Additional characteristics of these tools are the techniques used to produce mutants (via. Source code(SC) or byte code(BC) amendment) and techniques to do it fast like Mutant Schemata Generation (MSG) [21]. The mutant schemata generation method is used to create a single meta-mutant program that includes many mutants [3] at the source level and also the type of mutation testing that include strong mutation[7] or weak mutation [26] [27]. Weak mutation [26] [27] is an estimation method with which we can match the internal state of mutated program with the one it is created from exactly after executing the mutated part of the program. Very less computational cost/time is required to make sure that the test cases satisfy this type of mutation testing. On the other hand when the strong mutation [7] testing is executing, the values produced by mutated code are passed further and changes (or not) the output of the module or subroutine. If the mutant is not killed, then bearing in mind whether it can be killed by enhancing the test cases data or the live mutant cannot be killed by any test data. Strong mutation testing tells whether the code has to be like it is for the tests to pass. Moreover it requires much more computing power and the order of mutation whether first order or higher mutation order (HOM) [28]. Studies proved that the HOM [28] is better for checking the quality of test cases. The description about studied Mutation testing tools for various languages is provided as under;

Mothra

Mothra [23] was the first Mutation testing tool. It was implemented in C language to test the FORTRAN language code and developed at Georgia institute of technology. Mothra uses 22 types of mutant operators which are structured into 8 classes which are super structured into 3 levels statement analysis, coincidental correctness, predicate, and domain analysis. Mothra contains individual tools. Each tool performs independent functions for the software testing. Mothra works on source code. The tool contains a complete language system which takes the source code to be tested and translates it to the intermediate code so an interpreter can execute the mutated versions of the code [23]. 232

Proteum

PROTEUM(Program testing using Mutants) [13] is a tool of testing for C language. Development of this tool was done in Brazil at University of Sao Paulo (USP), Sao Carlos for research activities. Proteum can also be configured to test other procedural language. The Proteum uses 71 mutation operators to works with which are divided in 4 class categories; 1) Statement, 2) Operators, 3) Variables, and 4) Constants. Proteum imports 3 types of files 1) POKE TOOL test files, 2) ASCII files, and 3) other Proteum test case files. Insertion of test cases can be done in two ways, mainly by, initial parameters, and runtime input. Proteum can generate non executable mutants. Proteum has execution time constraint unit to discriminate mutants that can enter into an infinite loop [13].

Insure++

Insure++ [12] is a Mutation testing tool registered by Para Soft. Insure++ is a tool for C and C++. Insure++ employs a new approach of shifting the focus from the test cases directly to the main program in mutation testing. This approach of Insure++ enables it to find bugs that are present in the original source code. Insure++ creates equivalent mutants rather than creating faulty mutants. This approach makes Insure++ more effective for mutation testing during error detection, as if there is any type of equivalent mutant, it can be exposed beforehand. The patented Source Code Instrumentation (SCI) technology of Insure++ [12] gives detailed information of debugging, which makes bug detection and elimination very easy for the tester [12]. In the process of inspecting the bugs the Insure++ starts parsing, then analyzes it and then finally converts the source code to an equivalent code. A temporary file is created and the newly generated code is stored in it which is then compiled. The original source is not modified in Insure++. The process is completely automatic (user intervention is not required) and invisible to the user. Those equivalent mutants are run instead of the original code during the procedure of bug-detection. These 'equivalent' mutants are always expected to pass instead of fail, unlike faulty mutants. There is need of the fixture of the source code if any of the mutants is killed because the equivalent mutant cannot be detected. Insure++ is very useful in solving equivalent mutant problem which is considered to be a difficult task. Moreover, in Insure++ one can have desired number of mutants as an additional feature of the tool [12].

PlexTest

PlexTest [24] is a highly selective mutation tester. PlexTest is used to do mutation testing for C++ programs. PlexTest can only delete an instruction or some of its part e.g. it fully tests the binary operators in a program by deleting, along with the operator, part of the statement on left side firstly then the part of the statement on the right side of the operaor. PlexTest can implement both non-selective and indeed full selective mutation testing. PlexTest attempts to bridge the gap between coverage testing and traditional mutation testing. The tool makes sure that each statement is executed and tested verifiably with no issues in automatic nature of testing. [24].

Milu

Milu [11] is a Mutation testing tool for C language. It is designed to do both order of mutation testing. Milu has 77 C mutation operators in it. Milu adds a new feature in which a user can modify the set of operators to be applied. The customization can be done with the help of a scripting language known as Mutation Operator Constraint Script (MOCS). Milu also has a test harness technique to reduce the execution time for running the test cases. In this technique a test harness having all test cases and settings for running each mutant is created. [11].

Nester

Nester [14] is a freely available tool for Mutation testing for C# source code to extent the adequacy of the unit tests. It involves changes in the programs to check if existing test cases can discriminate the original program from the mutated one. Current version of Nester, (that is, 0.3 Alpha) holds only C# programs for the Microsoft Visual Studio 2005. At the point only NUnit framework is supported. Nester tool featured XML- based C# grammar and the corresponding parser. This allows specifying transformation rules based on grammar tokens [14].

PIT

PIT [19] is a fast byte code transformation based mutation testing system for Java which makes it possible to test the effectiveness of the unit tests. Unlike traditional 233 and branch coverage tools, PIT [19] ensures execution and detection of faults that are present in code by the tests. PIT not only confirms the execution of code by the tests, it also confirms the fault detection by the test cases. PIT has more than 35 mutants' that is divided into 13 classes. PIT produces easy to read HTML reports. PIT also performs the combining of line coverage and mutation coverage information [19].

MuJava

MuJava is a mutation testing tool and can be understood as a descendant to the JavaMut [18]. JavaMut [18] is not freely available for download and does separate compilation of each mutated class which seems to be a time consuming process and it does not execute the mutants according to Ma et al. [17]. To overcome and address the remaining problems [17], MuJava started with Chevalley and Thevenod Fosse's work [18] and hence regarded as the successor of JavaMut. MuJava offers [25, 26] a huge set of both the traditional mutation operators and OO mutation operators. MuJava is dedicated to the Java language. Additionally, MuJava implements two advanced techniques of 1) MSG and 2) Byte Code transformations aiming at decreasing time consumption in mutants generation. The MSG and byte code translation methods, executed in MuJava are considered to be quicker than to compile separately [17].

Judy

Judy is a tool of mutation testing build in Java with AspectJ extensions, which is implemented by the FAMTA Light (Fast aspect-oriented mutation testing algorithm) [45. 46]. The main and important features of Judyare to provide high performance, to integrate with the professional development environment tools, use of advance mechanism for mutant generation and to automate the process, which makes the tool to be used efficiently in industrial environment and in open- source projects. Judy supports latest version of Java that makes it to run testing process on latest components, the traditional mutation operators, namely, ABS, AOR, LCR, ROR, and UOI are also supported by Judy.

Bacterio

Bacterio [20] is a standalone Graphical User Interface (GUI) based application. Mutation analysis is implemented on java application by Bacterio and it supports test cases written in two types of test frameworks; JUnit and UISpec4J. Bacterio is a mutation testing tool that has the maximum number of features and techniques till now. Bacterio uses mutant reduction techniques. The desired number of mutant operators can be generated and test cases are executed on them. Bacterio uses different techniques for mutant generation and executions example Byte-code translation [26, 27], Mutant schema [21], MUSIC [29], Parallel executions [30]. Bacterio can also perform strong mutation [7] as well as weak mutation [42, 43] for execution. Additionally, new features of Bacterio include selection of tests based on mutation , explanatory mutation testing, and execution scripts [31, 32].

Jumble

Jumble speed up Mutation Testing by operating at a byte code level . Jumble is deployed in an industrial setting [16], and supports JUnit. Therefore, Jumble is worthy

of investigating further. Jumble supports limited set of operators for mutation testing. The simplified versions of AOR, ASR, COR, LOR, and SOR, studied in [25] are the mutation operators supported by Jumble. The available mutation operators are simplified such that 'minus sign' is switched with 'plus sign', and not by each of the other operators, that is, addition, subtraction, division, and 'percent' symbols as assumed by AOR. Additionally, jumble does not support convention that is the name adjustment of unit tests. Jumble has 5 types of mutation types, namely, 1) Conditional, 2) Arithmetic Operations, 3) Constants, 4) Return Value, and 5) Switch Statement, containing 20 operators [16].

Jester

Jester [15] is a tool used for Mutation testing and considered to be beneficial by some practitioners. Jester offers a way to enhance the available set of mutation operations. The problems regarding reliability and performance of the Jester tool and limited range of mutation operators is there which is based only on substitution of the string. Moreover, Jester does not offer context-aware mutation operators. Jester generates, then compiles it, and finally runs unit tests against a mutant, which is the approach being used by the tool. The process of mutants generation, compilation, testing, and reporting of the Jester is a problem because it reports for each mutant that is why it is little inefficient [15].

COMPARATIVE STUDY OF THE EXISTING TOOLS

All the compared Mutation Testing tools are available are standalone systems and some of them can be used as a plug-in also. The subsequent section shows the comparisons in the form of table and pie charts. Additionally, the various features for comparison among different tools for all languages include; Type of language the tools has been used for, Order of their Mutation, manually selection of mutation operators, mutation generation level, type of mutation testing, etc. and can be interpreted from the Table 1. Mutation testing tools supporting only Java language contains the features which are tabulated under Table 2.

Characteristic Features	Accessible Mutation Testing Tools											
	INSURE ++	JUMBLE	JESTER	NESTER	PIT	JUDY	MUJAVA	MILU	MOTHRA	PLEXTEST	PROTEUM	BACTERIO
Used for Language	C,C++	Java	Java	C#	Java	Java	Java	C	Fortran	C++	C	Java
Order of Mutation	First	First	First	First	First	First	First	Higher	First	First	First	Higher
Manually Selection of Mutation Operator	No	No	No	No	Yes	Yes	Yes	Yes	Yes	No	Yes	No

Table 1. Comparison between studied tools

Characteristic Features	Java Mutation Testing Tools					
	PIT	MUJAVA	JESTER	JUDY	JUMBLE	BACTERIO
Object-Oriented Mutation Operators	✓	✓	×	✓	×	✓
Type Of System	Standalone System and plug- in for eclipse.	Standalone system and plug- in for eclipse.	Standalone system	Standalone Application	Standalone system and plug-in for eclipse	Standalone system
Supported Java Version	Java1.5 and above	Java1.4 and above	Java1.4 and above	Java1.6 and above	Java1.4 and above	Java1.3 and above
Type of Mutation Testing	Weak	Weak	Strong	Strong	Weak	Weak
Level of Mutant Generation	SC	BC	SC	SC	BC	BC
Non-Executable Mutants production	✓	✓	✓	✓	✓	✓
Traditional Mutation Operators.	×	✓	×	✓	×	✓
Meta-Mutant	×	✓	×	✓	×	✓
Mutants Format	Cannot be seen In-memory	class files separately	Source files separately	Separate class file	In-memory cannot be seen	source files contains in groups
JUnit Support	✓	×	✓	✓	✓	✓

Table 2. Comparison between tools available for java

Comparison in the Tabular Form

The different tools available for mutation testing are studied and an empirical comparison among these Mutation testing tools is made as an inference both in the Tabular Form and in the form of Pie-Charts. Table 1 includes the tools available for both Procedural languages as well as Object-Oriented languages. Specifically for Java, i.e. Object-Oriented Language, Table 2 summarizes the features of Java Mutation testing tools. It can be concluded from the Table 1 that six tools out of twelve compared tools are used by Java language. These tools include MuJava, Bacterio, PIT, Jester, Jumble and Judy. Milu and Bacterio give higher order of mutation while other tools give first order of mutation. Half of the studied tools can perform manually selection of operators while others do not perform the manual selection of mutation operators. The Table 2 comprises of Java language supported mutation testing tools which includes; PIT, MuJava, Jester, Bacterio, Jumble and Judy. Table 2 gives an inference that MuJava, Bacterio and Judy supports Object- Oriented operators while other three tools, PIT, Jester and Jumble does not

support OO operators. TheMutation generation level of PIT, Jester and Judy is source code and others possess byte code. Additionally, all the tools produce non-executable mutants. Only Mu- Java, Bacterio and Judy support traditional mutation operators. Jester and Bacterio have strong type of mutation whereas others have weak mutation type. Maximum of the available tools are standalone type of systems. The Mutation testing tools, namely, MuJava, Jester and Jumble supports Java version 1.4 and above. PIT supports Java version 1.5, Bacterio supports 1.6 and Java version 1.3 and above is supported by Judy.

Pie-Charts Analysing Different Characteristics

This section is demonstrating different features of the Mutation Testing tools in the form of pie charts. The following pie charts are showing their distribution in this section. The figures 1 to 3 are showing distribution among the Mutation tools in terms of the language supported by them. Mutation Testing Tools available for different languages, namely, Java, FORTRAN, C, C++, and C# can be depicted from figure 1. The following figure 1 shows that half, i.e. 50 percent of the available tools support Java language. 16.67 percent shows number of tools for C and C++ languages while 8.33 percent of the mutation testing tools are available for C# and FORTRAN languages. The figure 1 depicts that the tools that supports Java language includes, MuJava, PIT, Jester, Jumble and Judy. Proteum, Milu and Insure++ are used for C language. Insure++ and PlexTest are used for C++ while Mothra is used for FORTRAN. Nester supports C#

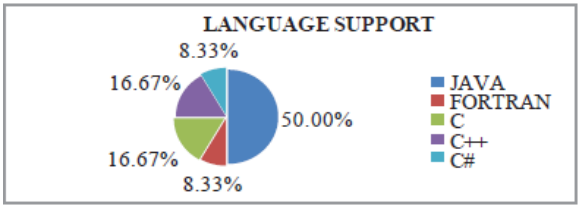

Fig 1. Distribution on supported language

language.

The following figure 2 is showing Order of Mutation. Except Bacterio and Milu which possess 16.67 percent of the total distribution for higher order utations, all other tools give first order mutation.

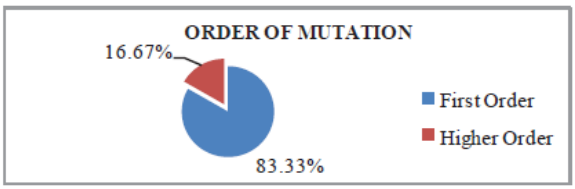

Fig 2. Order based distribution

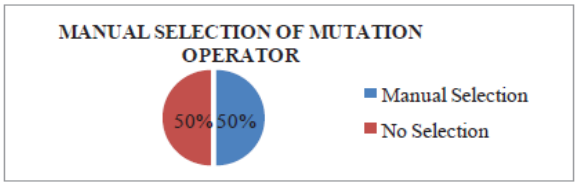

Fig 3. Manual selection of Mutation operator

The distribution based on manual selection of mutation operators is being provided in fig. 3. The Jester, Nester, Jumble, Judy, Insure++ and PlexTest does not provide manual selection of operators while the other half of the tools supports manual selection of operators. The following figures 4 to 11 are presenting distribution among the Java Supported Mutation testing tools. This distribution in the form of pie-charts is presented after comparing the features of Table 2. After analyses of different Java tools, it can be depicted from the following figure 4 that 50 percent of the tools are practiced as Standalone system, and 50 percent of the available tools are used both as standalone systems and as a plugin. Above figure 5 depicts the distribution on the basis of OO Mutation operators. MuJava, Bacterio and Judy comprises of 66.67 percent of the total distribution while others contribute 33.33 percent of the distribution.

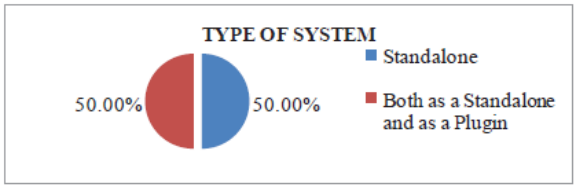

Fig 4. System Types

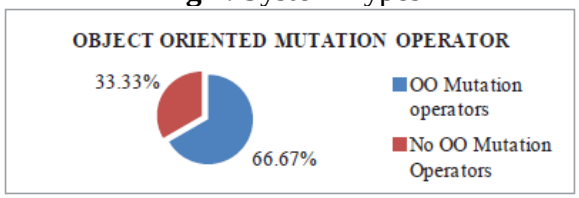

Fig 5. OO mutation operator distribution

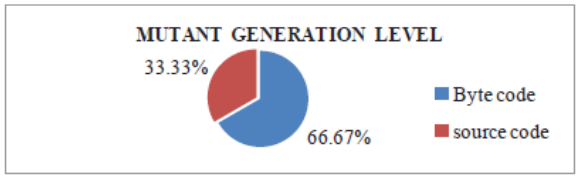

Fig 6. Mutant Generation Level based distribution

In the above figure 6, distribution is shown on the basis of Mutant Generation Level. PIT, Jester and Judy have source code generation level undertaking 33.33 percent of distribution. On the other hand, MuJava, Bacterio, Jumble has byte code mutation generation level and takes 66.67 of the total percentage.

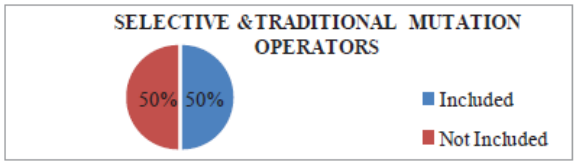

Fig 7. Distribution showing the Selective & Traditional Mutation Operators

The figure 7 is presenting distribution among various tools on the basis of selective and traditional mutation operators. The 50 percent of distribution is showing that half of the available tools include selective and traditional mutation operators while the rest 50 percent does not include these mutation operators. The meta-mutant support is shown by the above figure 8. Fifty percent of the distribution is showing that half of the available tools support meta-mutants while another 50 percent does not support the meta-mutants.

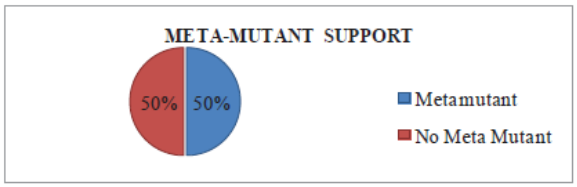

Fig 8. Distribution showing Meta-mutant support

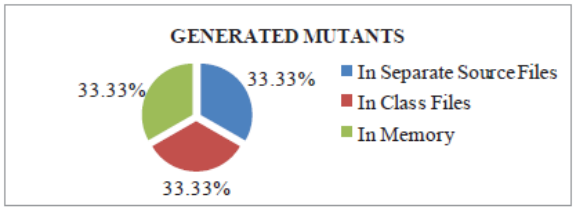

Fig 9. Distribution depicts the storage of mutants

Above figure 9 is indicating the distribution between mutation testing toos on the basis of generated mutants. 33.33 percent of the distribution is showing that generated mutants are stored in separate source files as can be interpreted from Table 2, 33.33 percent are stored in class files while the remaining 33.33 percent shows storage in the memory.

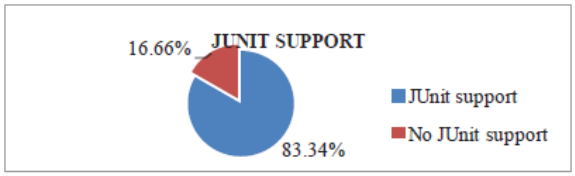

Fig 10. Distibution on the basis of JUnit support

The above figure 10 is showing distribution

among the Java tools that 83.34 percent of tools like PIT, Jester, Bacterio, Jumble and Judy are supporting JUnit whereas the remaining 16.66 percent of the distribution for MuJava does not support JUnit.

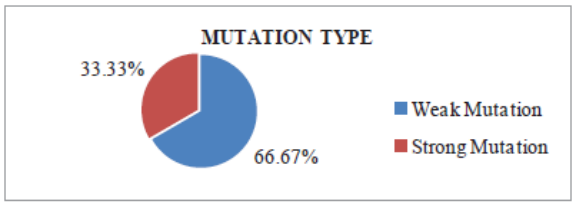

Fig 11. Distribution depicts the type of Mutation

Above figure 11 is depicting the distribution
among various tools available for Java language on the basis of Mutation type. Jester and Bacterio have
strong mutation type and comprises of 33.33 percent of the total distribution while the remaining 66.67 percent of the tools supports weak mutation type. The tools which have weak mutation type are PIT, MuJava, Jumble and Judy which can be interpreted from the Table 2 of previous section.

SUMMARY OF THE COMPARATIVE STUDY

It can be analysed from the empirical study and comparison that Java contains maximum number of mutation testing tools while other languages including FORTRAN, C, C++, C# have very few number of Mutation Testing tools. The studied tools determine that MILU and Bacterio have the feature to produce higher order mutation. The characteristic features of Java mutation testing tools in Table 2 shows that Bacterio, and Judy have many important advantages, making the tools efficient over other available tools. MuJava, Judy, PIT, and Bacterio support Object-Oriented Mutation Operator. Jester and Bacterio have strong type of mutation.

CONCLUSION AND FUTURE WORK

The comparative study shows that finding the effective test suits is pretty much easy and reliable through the Mutation testing yet there are some problems regarding the use of Mutation Testing because it consumed a lot of time and the cost of computation is very high.
Henceforth many automated tools for the generation
of mutants and test cases are available for performing
Mutation Testing on test cases. Many tools like MOTHRA, PROTEUM, and MILU are available but they were used by procedural type of languages. Similarly for Object Oriented language also that contains Java, C++, and C# languages the automated tools are developed. The mutation testing tools including MUJAVA, PIT, NESTER, JESTER, PLEXTEST, INSURE++, JUMBLE, JUDY and BACTERIO are the examples of above stated languages. All these tools are language dependent, which means they can generate and execute mutants for single languages. Therefore, it can be implicated from the comparative study that Java supports the maximum number of Mutation testing tools and other languages including C, C++, C# and FORTRAN have minimum number of the available tools for Mutation testing. The relevancy of the paper can be depicted from all the relevant references given in the end of paper. All the references give substantial support to the comparative study. This paper is the comparative study between the Mutation Testing tools for different languages including Java, C, C++ and FORTRAN in the market. Henceforth, the future work will include proposal of improving the performance of the tools and to build a new tool which can deal with most of the languages. An extension will be made as a new system which will be made

language independent after comparing all the market available tools.

REFERENCES

Andrews J.H., Briand L.C., Labiche Y.: 'Is mutation
an appropriate tool for testing experiments?' Proc. 27th Int. Conf. Software Engineering, May 2005, pp. 402–411.

[2] Andrews J.H., Briandl C., Labiche Y., Namina. S.:
'Using mutation analysis for assessing and comparing
testing coverage criteria', IEEE Trans. Softw.

[3] Ma Y.S., Kwon Y.R., Offutt J.: 'Inter-class mutation
operators for java'. Proc. 13th Int. Symp. Software
Reliability Engineering, November 2002, pp.
352–363.

[4] Offutt A.J., Lee A., Rothermel G., Untch R.H.,
Zapf C.: 'An experimental determination of sufficient
mutant operators', ACM Trans. Softw. Eng. Meth, 1996, 5, (2), pp. 99–118.
Eng., 2006, 32, (8), pp. 608–624.

[5]Wikipedia:
http://en.wikipedia.org/wiki/Mutation_ Testing

[6] Offutt A.J., Untch R.H.: 'Mutation 2000: uniting the
orthogonal', in Wong W.E. (ED.): 'Mutation testing
for the new century' (Kluwer Academic Publishers,
2001, 1st edn.), pp. 34–44.

[7] DeMillo R, Lipton R.J and Sayward F.G., Hints on test data selection: Help for the practicing programmer, IEEE Computer. 11(4): p. 34-41, 1978.

[8] M.P. Usaola, P.R. Mateo and B.P. Lamancha, "Reduction
of Test Suites Using Mutation", In 15th International
Conference on Fundamental Approaches
to Software Engineering. Tallin, Estonia, March,
2012.

[9] Walsh P.J.: 'A measure of test case completeness'.
PhD thesis, University New York, 1985.

[10] Wong W.E., Mathur A.P.: 'Reducing the cost of
mutation testing: an empirical study', J. Syst. Softw.,
1995, 31, (3), pp. 185–196.

[11] Jia Y, Harman M, MILU: A Customizable,
Runtime-Optimized Higher Order Mutation Testing
Tool for the Full C Language, Testing: Academic
& Industrial Conference - Practice and Research
Techniques.

[12]
http://www.parasoft.com/jsp/products/article.jsp?
articleId=291#a

[13] Delamaro M.E., Maldonado J.C.: 'Proteum– a
tool for the assessment of test adequacy for C
programs'. Proc. Conf. Performability in Computing
Systems, July 1996, pp. 75–95.

[14] http://nester.sourceforge.net

[15] Moore I.: 'Jester – a JUnit test tester'. Proc. Second
Int. Conf. Extreme Programming and Flexible
Processes in Software Engineering, May
2001, pp. 84–87.

[16] Irvine S.A., Tin P., Trigg L., Clearyj.G, Inglis
S., and Uttingm: 'Jumble Java byte code to measure sharp

the effectiveness of unit tests'. Proc. Testing:
Academic and Industrial Conf. Practice and Research
Techniques, September 2007, pp. 169–175.
[17] Ma Y., Offutt J., Kwon Y.R., MuJava: an automated
class mutation system, Softw. Test Verif. Rel.,
2005, 15, (2), pp. 97–133.
[18] Chevalley P., Thevenod-Fosse P., A mutation analysis
tool for Java programs, Int. J. Softw. Tools
Tech. Transfer, 2003, 5, (1), pp. 90–103.
[19] http://pitest.org/
[20] Pedro Reales Mateo, Macario Polo Usaola Bacterio:
Java Mutation Testing Tool A Framework to Evaluate Quality of Tests Cases 2012 28th IEEE International
Conference on Software Maintenance (ICSM).
[21] Untch R.H., Offutt A.J., Harrold M.J., Mutation
analysis using mutant schemata, Proc. Int. Symp.
Software Testing and Analysis, June 1993, pp. 139–
148.
[22] Ma Y.S., Harrold M.J., Kwon Y.R.: 'Evaluation of
mutation testing for object-oriented programs'.
Proc. 28th Int. Conf. Software Engineering May
2006.
[23] King K.N., Offutt J., A FORTRAN Language System
for Mutation Based Software Testing, Software—
practice and experience, vol. 21(7), 685–718 (July 1991).
[24]
http://www.itregister.com.au/products/plextest_
detail.htm
239
[25] Ammann P., Offutt J., Introduction to software
testing (Cambridge University Press, 2008, 1st edition).
[26] W.E. Howden, "Weak Mutation Testing and Completeness
of Test Sets," IEEE Trans. Software Eng.,
vol. 8, no. 4, pp. 371-379, July 1982.
[27] A.J. Offutt and S.D. Lee, "An Empirical Evaluation
of Weak Mutation", IEEE Transactions on Software
Engineering. 20(5): p. 337-344, 1994.
[28] Y. Jia and M. Harman, "Constructing Subtle Faults
Using Higher Order Mutation Testing," Proc. Eighth
Int'l Working Conf. Source Code Analysis and Manipulation,
pp. 249-258, Sept. 2008.
[29] P.R. Mateo and M.P. Usaola, "Mutant Execution
Cost Reduction, through MUSIC (MUtant Schema
Improved with extra Code)", In IEEE Fifth International
Conference on Software Testing, Verification
and Validation. Montreal, Canada, 17th April, 2012.
[30] P.R. Mateo and M.P. Usaola, "Parallel Mutation
Testing", Software Testing, Verification and Reliability.
2012.