I.J. Modern Education and Computer Science, **2014, 11, 1-10**
Published Online November 2014 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijmecs.2014.11.01

A Study and Review on the Development of Mutation Testing Tools for Java and Aspect-J Programs

Pradeep Kumar Singh
Department of Computer Science Engineering, Amity University, Uttar Pradesh, India
Email: pradeep_84cs@yahoo.com

Om Prakash Sangwan
School of ICT, Gautam Buddha University, Greater Noida, India
Email: sangwan_op@yahoo.co.in

Arun Sharma
Indira Gandhi Delhi Technical University for Women, New Delhi, India
Email: arunsharma2303@gmail.com

Abstract—Mutation analysis in software testing is observed as the most effective way to validate the software under inspection. In last decade, number of researchers developed various methods and tools to apply mutation testing on Aspect Oriented Programs. In this paper, authors analyzed numerous mutation testing based tools available to test the Java and AspectJ programs. All effective and popular Aspect-J testing tools have been considered and analyzed in this paper, based on essential requirements in this context, considered to be fulfilled by testing tools decided by testing professional and researchers for such tools. This paper analyzed the work progress in the field of mutation testing techniques and tools specific to Java and AspectJ. This work considered essential parameters on which the analysis of analyzed tools is carried out. In case of addition parameters considered for evaluation, some of the resultant metrics may vary slightly under modification in basic requirements. Based on the numeric value estimated, it is finally suggested the merits of a mutation tool under different circumstances. This is the extension of the work carried by us in previous review for aspect based mutation testing techniques.

Index Terms—Software Testing, Mutation Testing, Aspect Oriented Programs, Mutation Testing Tool, ITDs: Inter-type Declarations, Join Points, Pointcut Descriptors and Mutation Analysis.

I. Introduction

The A significant proportion of the total cost of software is attributed to testing over its lifetime [1]. One way to reduce this cost is to increase software testability. Testability is a measure of how easily software exposes faults when tested [2]. By improving testability the cost of testing is reduced. Aspect-Oriented Programming (AOP) [3] support programmers in identification of separation of concerns: separate a program into distinct parts that overlap in functionality. AOP provide better modularity in programs, which is basic prerequisite in software engineering discipline. It can also reduce the development effort, testing time and provide better reusability and maintenance [35] as compare to OOP in several aspects. The main constituents of AOP languages are aspects, pointcuts, joinpoints, and advices. The encapsulation of joinpoint, pointcut, and advice is called an aspect. Pointcuts is the code that match the joinpoints, which perform a specific action on call called advice. Advice contains its own set of rules as to when it is to be invoked in relation to the joinpoint that has been triggered. Joinpoints are pre-defined locations inside the source code where a concern will crosscut the application [1]. Joinpoints can be method calls, constructor invocations, or some other points in the execution of a program. The crosscutting behavior of AspectJ can be partitioned into two major sections: in term of the behavior (advice) and applicability of the behavior (pointcut)[1,4].

There are several testing techniques proposed by many authors to test the program written in aspect oriented languages like aspect-J, Hyper-J etc. In this paper major emphasize is given on Java and Aspect-J because of the limited scope of the research paper. Most of the researchers worked on Aspect-J because it is having well defined constructs of language that makes it easier to use as compare to other AOP languages like Cease-J, Hyper-J. We also believe that readers interested in mutation based testing tools can use this paper as a road map to analyze the strength of each tool and technique discussed in this survey. This review gives an empirical evaluation based on the various parameters considered or validated and reveals those area that are not addressed or areas that require further research. This paper is the extension of the work carried by us in previous review for aspect based mutation testing techniques and tools [52].

Copyright © 2014 MECS

This paper is arranged as follows: Section II provides basic research procedure used to conduct this survey. Section III reports the detail of fault based mutation testing in Java and AspectJ respectively. Major findings with exhaustive literature review on mutation tools for Java and AspectJ is presented in Section IV with conclusion in Section V.

II. Research Procedure

Defining an adequate search string is quiet difficult for analysis. Most of times identify the string for search relies on the experience of the involved researchers. According to our research survey, we defined the following string:

(Aspect-oriented programming OR aspect-oriented application OR aspect-oriented program OR AOP OR Java) AND (mutation testing OR fault based testing OR testing in AOP OR mutation tools OR testing tool) AND (Mutation Tools for Java)

The sources of primary studies vary from indexed repositories (IEEE, ACM Digital Library, Elsevier, Science Direct, ICST, IJCSE, ICIIP, IJCA etc) to general purpose search engines(Google and Scirus). We have downloaded 250 papers out of which, we have considered eleven papers for mutation tools in Java, four paper for mutation testing techniques and six tools for AspectJ programs during analysis. We have limited to our findings for mutation testing techniques and tools because it is not feasible to analyze all the tools in one paper, so we restrict our finding specific to Java and AspectJ programs based on mutation testing. However, use of mutation testing among software professionals in software industry is rare because of unavailability of automated tools and running cost and time of vast numbers of mutants against the test cases. Now a day's, tools are gaining more popularity because of the effectiveness and efficiency over mutation testing techniques makes strong ground to do analysis.

In this paper we have considered most of the basic requirements for developing the testing tool given by various researchers in their research work. We have assigned a weight value based on the possible requirement fulfilled by the given testing tools for AspectJ. Here the reference point is the information available in research papers. Numerical values assigned to requirements are based on empirical data reported in the published research papers for the mutation tools in AOP as follows:

Yes=1, Partial=0.5, No=0

III. Fault Based Mutation Testing

For Mutation testing computes how good our tests are by injecting faults into the program under test. Mutation testing is fault-based testing method that estimates the effectiveness of test cases [7]. Mutants are faulty kind of program which contains some faults. The mutants are generated from the original program by applying changes to its original code (e.g. k + 21 → k -21). Each modification is examined by a mutation operator. To detect the faults, test cases are then used to check whether the mutants and the original program produce dissimilar responses or not. The number of mutants identified provides a measure of the quality of the test suite called mutation score. However, the standard of the mutation testing depend upon the quality of the mutation operators, which must reveal realistic fault types [14].

A. Mutation Testing

Mutation testing [7, 51, 52] became popular among the testing researchers for over 25 years and is conventionally used to measure the efficacy of test suites. Moreover, mutation provides a comparative technique for inspecting and upgrading multiple test suites. A number of empirical investigations (e.g., [11, 29]) have relied on using mutation as part of the experimental process.

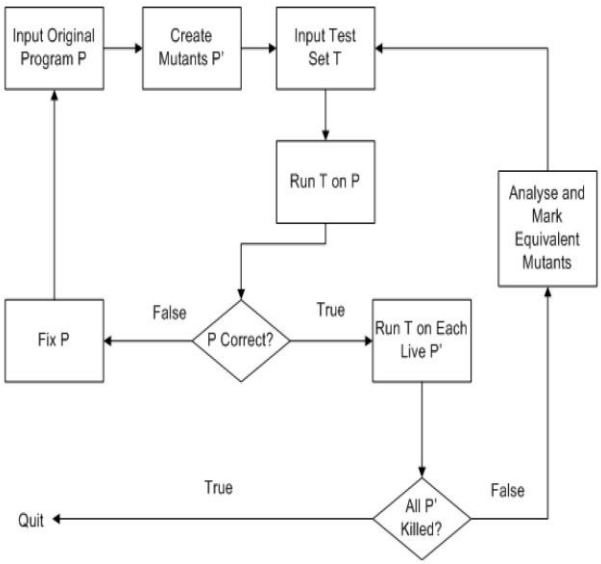

Fig.1Generic Process of Mutation Analysis [36, 50]

Mutation testing finds the adequacy that tests detects all mutant or not. Mutation testing comprises many costs, including the possible generation of vast numbers of mutants. Another cost of mutation testing is the identification of equivalent mutants [1, 4]. Equivalent mutants, by definition, are unkillable because the mutants are semantically similar to the original program. Recognizing such mutants in software is generally intractable [1, 9] and historically has been done by hand [1, 8]. Till now, significant work has not been done to carried out the automatic detection of the equivalent mutants and some software practitioners also suggested to discard the equivalent mutants under few circumstances.

There are several steps which are associated while applying mutation testing in general or on Aspect Oriented Programming. The process of traditional mutation testing started with constructing the mutants of

a test program [9, 10]. The detailed testing process is graphically shown in Fig. 1.

B. Mutation Testing in AOP

Amending in AOP does wide influence on dynamic behavior of affected program, which makes testing harder in AOP compared to OOP (Object Oriented Programs). Fault based testing is testing technique for Aspect Oriented programs based on the classification of fault types and then insertion of faults in the Aspect Oriented programs. We have taken fault based mutation testing tools into account for analysis.

Fault based testing was introduced by Zhao and Alexander [5] in 2007, they discussed a process for the efficient testing of aspect oriented program on the basis of different faults. Before the existence of this technique Alexander *et al.* [6] identified faults classification for Aspect Oriented programs. It was the first work done on fault identification for AO programs. Author presented the list of different faults specifically for AO programs and then on the basis of those faults he constitutes a fault model for AO program [6]. There are three elements in a complete AspectJ fault model presented by Zhao *et al.* [5], 1) fault model for pointcuts, advice, intertype declaration and aspects 2) fault lists for pointcuts, advice, intertype declaration and aspects 3) java fault model which contains the java related faults. All three elements are necessary for the working of fault based testing technique on AO programs [5].

In [5], for effective and efficient use of Fault Based testing authors have explained dependence model and interaction model for weaved base and AO code. Complete structure of the fault model presented by [5] has been shown in Fig. 1, which can be used for fault based testing of AO programs. Alexander et al. [6] recommended, a fault model for aspect-oriented programs, which includes six categories of faults: (1) Incorrect strength in pointcut patterns; (2) Incorrect aspect precedence; (3) Failure to establish post conditions; (4) Failure to preserve state invariants; (5)Incorrect focus of control flow; (6)Incorrect changes in control dependencies.

Ceccato *et al.* [35] extended the fault model derived by Alexander *et al.* [6] with three new fault categories: (1) Incorrect modifications in exceptional control flow; (2) Failures due to inter-type declarations; (3) Incorrect modifications in polymorphic calls. Fault based testing in AOP demands sound knowledge of behavioural and syntactical interactive relationships of software programming paradigms. This helps a software tester to identify different categories of faults which is beneficial for the systematic testing of software and getting intended results.

Mutation testing in AOP is a variety of fault based testing. The mutation testing of pointcuts is performed in two ways: (1) By creating effective mutants of a pointcut expression; (2) Testing the mutants using the designed test data. In [12], Prasanth et al. have introduced the mutation testing technique for pointcuts in AO programs. Lemos *et al.* [13] have used mutation testing to identify a fault type related to pointcut descriptors (PCD) introduced by Alexander *et al.* [6].

In [13], authors have first explained that while integration it is common that unintended and intended joinpoints are selected which causes problems in further execution of program and for testing of programs. They have used mutation testing to solve the problem of unselecting intended joinpoints. Whereas, according to Anabalagan *et al.* [12], during AOP software development sometime it happens that more than required or less than needed pointcuts are selected by software developers. Due to the use of wildcards usually a large number of mutants of pointcuts are selected for the testing purposes and it is difficult to select the most relevant mutants among a large number. In this situation it is problematic for software testers to efficiently design the test suite for that particular software. According to authors with the use of their technique it is possible to select the closely similar mutants for the mutation testing suite [12]. Mutants can be categorized according to the pointcut, advice and intertype declarations (ITDs). Principally four types of mutation operators available namely pointcut, advice, waving and base programs related operators.

IV. Literature Review On Mutation Tools

In this section total 11 mutation tools for Java and 6 tools for AspectJ programs were taken for study and analysis.

A. Literature Review on Mutation Tools for Java

There are number of testing tools available for object oriented programs. But most work is related to the family of C++ or Java Programs. In this study we considered only Java related testing tools specific to mutation testing. In this study total eleven tools were analyzed for Java and six for AspectJ.

A.1 Jester, it can be considered as the first mutation tool for Java programs. It is an open source tool. It builds some modifications to the code, runs the build (run tests) and if the tests pass Jester displays a message saying what it modified (Mutation Testing). Its detail implementation is available on the URL [37].

A.2 JavaMut, introduce the first prototype graphical user interface (GUI) based tool assisting mutation analysis of Java programs. The JavaMut tool was developed in the framework on elucidating a test strategy for critical avionics systems implemented in Java [38]. The prime objective was to automatically run large campaigns of mutation analysis experiments to analyze the strengths and weaknesses of test cases plot from UML state diagrams. It supports a GUI that helps the tester to customize mutation analysis. Three interfaces were provided to tester to generate mutants, to identify equivalent mutants, and to visualize statistics particulars. It is used on UNIX workstations for the detailed examination of two mid-size case studies related to dissimilar domains: an avionics application and a banking application.

Copyright © 2014 MECS

A.3 MuJava is the outcome of a joint work between Korea Advanced Institute of Science and Technology (KAIST) in South Korea and George Mason University in the USA. MuJava is now accessible for experimental and educational use. Information related to the tool, and comprehensive instructions as to how to install and use MuJava are available at the URL [39, 40, 41]. MuJava has three primarily roles: (a) generating mutants, (b) analyzing mutants, and (c) running the test cases supplied by the tester. MuJava uses the class com.sun.tools.javac.Main included in JDK to compile mutants [39-41].

Testers provide the test cases in form of methods that have sequence of calls to methods in the given class. To measure outputs of mutants with outputs of the original class, considered test method should have no parameters and return a string result. MuJava provides a GUI. Main benefit of this approach is that it demands only two compilations: compilation of the original source code and compilation of the MSG meta mutant [39-41]. This highly minimizes the time required for mutant creation. Another use of the MSG/bytecode translation is portability. The recent version of MuJava has few usability issues. Till now, it does not display mutants in a very suitable way.

A.4 ExMAn is an automated and flexible mutation tool for Java programs. It support different quality assurance techniques such as testing, model checking, and static analysis [42]. Its prime objective is to allow automatic mutation analysis. The ExMAn architecture is constituted of three types of components: built-in components, plug-in components, and external tool components. The flexibility of ExMAn exists because of the distinct built-in components that can be used in any mutation analysis. Major limitations with ExMAn are: (a) Limitation in adding semi-automatically or automatically identifying equivalent mutants (b) Lack of automatically specify patterns for the creation of mutation operators (c) Expansion of the selected artifacts to allow for the selection of multiple quality artifact sets for each type and thus allow for statistical analysis.

A.5 MUGAMMA is technique that determines whether user's executions would have killed mutants and if so, apprehends the state information about those executions [43]. In this paper, author's represented a novel technique that simplifies performing mutation testing. MUGAMMA is a specialization of the GAMMA framework—it creates the instrumented versions of a program, and manages the deployment of the system and describing the results. Even with the difference in the number of mutants generated, MUGAMMA is superior to MUJAVA in context of the time to generate the mutants. In the implementation of MUGAMA, the use of MSG method improves the performance. But more analysis is required to validate the results because number of mutants generated by MUGAMMA was lesser as compare to MuJava because of the limitation of the tool to conventional mutant. There are some limitations as this paper describes prototype implementation of MUGAMMA. Prototype implements for MUGAMA only the selective set of conventional mutants for Java. They mentioned in their work that currently extending MUGAMMA to include class mutants. Comparison of the mutant-generation phase of MUGAMMA with the mutant-generation phase of MUJAVA is carried out with two examples.

A.6 MuClipse is the extension to MuJava. MuClipse supports Eclipse plug in and integration with the development environment, which both generates and tests mutants in Java 1.4 [44]. In producing mutants, MuClipse authorize the developer to select the mutant operators to select and which classes should be mutated. With the help of Eclipse View, it shows each mutant and its status, organized by Java class and producing operator [44]. It also presents the overall data for live and killed mutants, and finally the measured mutation score. Two open source projects were analyzed through this tool for validating the results.

A.7 Jumble is a class level mutation testing tool that mutates a class at the byte code level and executes its respective unit test to measure the number of killed mutants [45]. If all test passes, the mutant lives and result is tabulated. Similar to MuJava, just one mutation is possible at a time, over the source code under test. First, the tool runs all the tests on the original, unmodified, source file and checks whether they pass or not, recording the time necessary for each test. Then, it mutates the file according to different mutations operators and runs the tests again. It returns a mutation score with details regarding each live mutant. The process is done when all the mutations have been tested. It supports JUnit 3 and, recently, it was updated to work with JUnit 4.

A.8 Testooj is a testing tool, developed in Java, for testing Java programs. It allows two main functionalities (i) creation of test cases based on regular expressions (R.E.) (ii) execution of test cases to perform different types of result analysis [46]. Testooj is a useful tool, easy-to-use. The test case generation functionality is appropriated both for practitioners and for researchers.

A.9 Javalanche work on byte code. It evaluates the effect of individual mutants to effectively remove equivalent mutants. This tool has been demonstrated to work on programs with up to 100 KLOC. It supports a unique feature that it ranks mutations by their impact on the behavior of program functions [47, 48]. Higher the weight of an undetected mutation, the lower the chances of the mutation being equivalent (i.e., a false positive)—and the more the chances of undetected i.e. a serious defects. It overcomes two main problems with mutation testing: efficiency and equivalent mutant's problem.

A.10 In [31], Madeyski *et al.* proposed a mutation testing tool called Judy. It takes advantage of a novel separation of concerns mechanism, to avoid multiple compilations of mutants and, therefore, it help in speed up mutation testing. An empirical investigation of Judy with MuJava tool on 24 open-source projects have been demonstrates. It is based on enforcement of the FAMTA Light approach developed in Java with AspectJ extensions. The main characteristics of Judy are as follows (i) high performance (ii) advanced mutant

Copyright © 2014 MECS

generation technique (iii) integration with professional development environment tools (iv) full automation of mutation testing process and support for the latest version of Java (v) Allow it to run mutation testing against the most recent Java software systems or components.

A.11 MAJOR is a fault seeding and mutation investigation tool that is integrated into the Java [49]. It also minimize the mutant generation time and enables efficient mutation analysis. It has already been successfully validated on large applications with up to 373 KLOC and 406,000 mutants. Moreover, MAJOR's domain specific language support for specifying and adapting mutation operators also makes it extensible. Due to its ease-of-use, efficiency, and extensibility, it is an absolute tool for the study and application of mutation analysis.

MAJOR is a full mutation analysis framework, which supports strong and weak mutation. It consists of the two main parts (a) Mutation component: integrated in the Java compiler (b) Analysis component: integrated in Apache Ant's JUnit task. A binary version of MAJOR for Java 6 is obtained for analysis and use. There are some limitations such as, Implementation of new mutation operators, comparison with related tools, and Integration of conditional mutation into a C/C++ compiler. All analyzed tools are reported in Table I as shown below.

Table 1. Testing Tools for Java

S. N.	Name	Authors	Year	Technique	Available
1	Jester	Moore *et al.*	2001	General	Y
2	JavaMut	Chevalley *et al.*	2002	General	Y
3	MuJava	Ma *et al.*	2004	MSG and Reflection Technique	Y
4	ExMAn	Bradbury *et al.*	2006	TXL	Y
5	MUGAMMA	Kim *et al.*	2006	Remote Mutation Testing Technique	Y
6	MuClipse	Smith *et al.*	2007	Weak Mutation, Mutant Schemata, Eclipse Plug-in	Y
7	Jumble	----	2007	General	Y
8	Testooj	Polo *et al.*	2007	Regular Expressions	Y
9	Javalanche	Schuler *et al.*	2009	Invariant and Impact Analysis	Y
10	Judy	Madeyski *et al.*	2010	FAMTA	Y
11	MAJOR	Just *et al.*	2011	General	N

B. Literature Review on Mutation Tools for AspectJ

Firstly, we have reviewed the literature based on mutation testing techniques and secondly, mutation testing tools. A brief summary of the mutation testing tools for Java is mentioned in Table I, techniques and tools with respect to aspect programs have been shown in Table II and Table III respectively. Evaluation of the different mutation testing techniques and tools for aspect oriented programs based on various parameters mentioned in above mentioned tables. There are several testing techniques like unit testing, data flow based testing and their related tools are available in AOP. Mutation testing considered here for analysis because of mutation technique effectiveness and its strength in covering the most of faults in software programs.

Ferrari *et al.* [14] identified the AO fault type and proposed some set of mutation operator. Here, the faults are scattered in four sections related to:(F1) pointcut expressions; (F2) ITDs and other declarations; (F3) advice definitions, implementations and (F4) the base program. They have mentioned three new fault types in this study which were not included previously. However, a full and refined analysis regarding all AO implementations is out of the scope of this paper. Cost analysis for application of the operators based on two real world applications have been demostrated.

In [12], Anbalagan *et al.* recognized the automatic generation of mutants for a pointcut expression and identification of mutants that are similar to the original expression. Software professionals may use the test data for the woven classes against these mutants to apply the mutation testing. Framework discussed, serves two objectives; generating relevant mutants and detecting equivalent mutants [12]. Later, the framework also minimizes the total number of mutants from the initially generated mutants. For classifying the mutant for selection, original point cuts are compared with mutants and their matched joinpoints. Identification of mutant is based on Fault Model given by Alexander et al. [15] and as well as AJTE (AspectJ Testing Environment)[16]. This framework reduces software professional's efforts in identification and generation of equivalent mutants. Authors demonstrate the preliminary experiments on a few sample sets of an AspectJ benchmark called Tetris [17]. In implementation, join points are marked based on static analysis of the code. Currently this framework does not support dynamic context.

Fault-based testing to aspect-oriented programs i.e. AspectJ programs, using both coverage and mutation techniques are proposed by Mortensen *et al.* [15]. Mortensen et al. proposed a set of coverage criteria as well as guidelines for aspect structure. In this work, lack of validation to mutation operators to see if it corresponds to real faults or not is missing. Mutants were created in an ad-hoc manner; automatic mutation of pointcuts needs more investigation. Future work in this direction may development of an integrated set of tools to analyze AspectJ programs, gather coverage criteria, and generate and test program mutants.

Copyright © 2014 MECS

Singh *et al.* [18] surveyed various papers to identify the classification of various fault types for Aspect Oriented programs. Based on the analysis of considered paper they concluded that majority of faults occurs because of base program and aspect code. Finally they have listed some new fault types for AspectJ. But neither theoretical nor the empirical evaluation is given. This classification is not even verified with any application built in aspect language.

Table 2. Analysis of Mutation Testing Technique for Aspect Oriented Programs

	Parameters	Rashid *et al.* [14]	Anbalagan *et al.* [12]	Mortensen *et al*. [15]	Singh *et al*. [18]
	Source	IEEE	IEEE	Workshop on AOP	IJCSE
1.	Theoretical/Empirical	Theoretical	Empirical	Theoretical	Theoretical
2.	No. of Mutants	NA	Mutants Numbers 1445(2)	NA	NA
3.	Mutation Score	Not Considered	Automatic	Manual	Manual
4.	Future Extension/ No Future Extension?	Yes	Yes	Yes(Tool Support)	Yes
5.	Based on OO Testing or not?	No	Yes	No	No
6.	Model/Framework/Technique	Three new fault types and cost analysis on real world application.	Framework	Technique	Framework (added few types of mutants)
7.	Technique/Framework is validated or not?	Cost Analysis for fault type given	Preliminary experiment is carried out.	Validation is missing.	Not Validated
8.	Major Contribution	Fault Type and Propose some set of mutation operators	Fault Type Identification	Coverage Criteria	Fault Identification

As observed by Ferrari *et al*. [14, 28], the PCD is the location that is the maximum fault-prone in an aspect. Pointcut descriptors in aspects are crucial because they specify the locations where a concern should be woven. A test-driven approach for the development and validation of the PCD is proposed by Delamar et al. [19]. Authors developed a tool, Advice Tracer which enriches the JUnit API with new types of assertions that can be used to identify the expected joinpoints. Advice Tracer [20] allows a programmer to write test cases that focus on checking whether or not a joinpoint has been mapped by the PCD. Tetris and Auction applications were used as an example with three and two aspects respectively. Authors performed analysis to measure the effectiveness of the test cases written using AdviceTracer in terms of their effectiveness to detect faults bring by AjMutator in the PCD.

Anbalagan *et al.* [4] introduces an APTE (Automated Pointcut Testing for AspectJ Program), an automated suite that tests pointcuts in AspectJ with AJTE. This new APTE suite recognizes joinpoints that match a pointcut expression and a set of boundary joinpoints. In the target classes, this suite output the list of matched joinpoints. Authors implemented the framework for AspectJ and Java code using the Byte Code Engineering Library (BCEL) [21], Java reflection API [22], and AJTE. The examined version supports an AspectJ compiler called ajc [23] Version 1.5 and Java 5 [24]. The main components of the suites comprises the test bench generator, pointcut generator, candidate generator, and distance measure component.

Cistron, is the another mutation tool reported by Singh *et al.* [25] for Aspect J programs. It is based on testing tool MuJava. It implements most of the fault types and

generates automatic mutants. Cistron produced mutant code automatically on the basis of mutation operators. Authors applied testing on selected mutants to reduce cost of testing. It also identifies the equivalent and non equivalent mutants from live mutants. Results for generated mutant, initial test cases and added test cases are shown for three AspectJ applications. This tool will generate test data with more accuracy and more statistical analysis are mentioned in future work.

Testability can be measured through mutation analysis. In mutation analysis, a mutation tool generates faults for locations in software. Testability of a location is measured by executing tests against mutants and counting the proportion of mutants that cause test failure. To quantify the testability we required, mutant generation tools. Jackson *et al.* [26] introduced MuAspectJ, a tool for generating mutants for AspectJ programs. MuAspectJ tool is evaluated in terms of the quality of mutants it generates. The tool is evaluated in terms of how fast mutants can be generated and executed. This does provide some sense of the length of time that it will take to get to a result but does not provide any indication of the quality of generated mutants. This type of evaluation does not however provide any sense of how the mutants will impact on the assertions that can be made from analyzing the results. Although speed of generation and execution are practical issues that must be considered when performing mutation analysis, that can be easily addressed through parallel execution of mutants in a distributed mutant execution approach. Two main contributions of this tool are firstly, the provision of the MuAspectJ that can be used to generate mutants for AspectJ programs and secondly, the introduction of location coverage and mutation density as a means to measure the quality of generated mutants.

A tool, AjMutator for mutation analysis of PCDs is presented by Delamare *et al.* [27]. AjMutator separate the mutants according to the set of joinpoints found similar compared to the set of joinpoints compared from initial PCD. For a particular class of PCDs, automatic grouping result to equivalent mutants. AjMutator may also execute the set of test cases on the mutants to give a mutation score.

AjMutator is constituted in three ways; building of mutant source files from AspectJ source file, compilation of the mutant source files & execution of test cases on the mutants to evaluate the mutation score for the particular set of test cases. It is capable of generating and compiling large number of mutants on large systems. Manual preference of the mutants would have been time consuming and difficult. So the automatic classification of mutants by this tool, offers extreme benefits. Generating large numbers of mutant automatic classifications seems to be better. The automatic classification of the equivalent mutants also eliminates the useless execution of these identified mutants and offers a precise mutation score.

Ferrari *et al.* [28] presented a tool, named Proteum/AJ, which automates the mutation testing for AspectJ programs. Proteum/ AJ helps in primary steps of mutation testing approach and fix number of requirements for mutation-based testing tools such as mutant handling, test case handling and mutant analysis. A set of mutation operators and supports in meeting mutation testing criteria such as program execution, mutant generation, mutant execution and mutant analysis is provided by this tool [7].

In this paper, basic requirement for developing mutation testing tool based is considered for analysis [28, 30-34]. Based on literature available we have considered most of the essential requirements of mutation based testing tools for AspectJ programs. Considered requirements are elaborated as follows:

1) Test Case Handling: It concerns the execution of test cases and their activation or deactivation.
2) Mutant Handling: It deals with the creation of mutants, selection of mutants, execution and evaluation of mutants.
3) Adequacy Analysis: It covers the calculation of mutation score based on equivalent mutants, dead mutants and total used mutants.
4) Reducing Test Cost: Testing cost is reduced or not?
5) Unrestricted Program Size: It is related to the program size considered for testing.
6) Support for Test Strategies: It analyze whether the order of mutation operators to apply on the selected software is allowed or not.
7) Independent Test Configuration: Recognize that test input and output is confined by the tool.
8) Test Case Editing: It considers the changes in existing test cases or refinement of available test cases.
9) Automatic Program Execution: It considers the execution of actual programs as well as mutants. Spot that program should execute or compiles automatically.
10) Evolution of Equivalent Mutant: This is related to the creation of equivalent mutants and uses a technique to record the equivalent mutants.
11) Multiple Language Programs Support: This is related to the support various aspect programs written in different language.

Based on above mentioned requirements an analysis is carried for evaluation of mutation testing tools for Aspect Programs. We conduct a survey based 20 software testing professionals from Industry and Academics, which was not carried in our previous study reported in [52]. Priority is assigned to requirements because each requirement is considered dissimilar based on expert opinion. Importance assigned to each requirement is between 0 and 1(1; considered as utmost important and 0; as less important) to the testing tools for Aspect-J programs. Here most of the tools reported for Aspect-J because of the availability of data on it and contribution of research in this context. Detail analysis is shown in Table III as follows:

Copyright © 2014 MECS

Table 3. Evaluation of Mutation Testing Tools for Aspect-J

Mutation Testing Tools/Framework							
Requirements		Advice Tracer[19]	Anbalagan & Xie[4]	Cistron [25]	Mu-AspectJ[26]	Aj-Mutator [27]	Proteum /AJ[28]
Source		ICST 2009	IEEE	ICIIP 2011	ACM	IEEE	ACM
Model/Tool/ Framework	Rank	Approach	Framework	Mutation Tool	Mutation Tool	Mutation Tool	Mutation Tool
1.Test Case Handling*	0.7	0.35	0.0	0.35	0.35	0.35	0.35
2.Mutant Handling*	0.8	0.0	0.40	0.40	0.40	0.40	1.0
3. Adequacy Analysis*	0.7	0.35	0.35	0.35	0.35	0.35	0.35
4. Reducing Test Costs *	0.8	NA	0.0	0.0	NA	0.0	0.0
5.Unrestricted Program Size**	1.0	0.0	0.0	0.0	0.0	0.0	1.0
6.Support for Test Strategies***	0.7	0.0	0.0	0.0	0.0	0.0	0.35
7.Independent Test configuration	0.6	0.0	0.0	0.0	0.0	0.6	0.6
8.Test Cases Editing	0.7	0.0	0.0	0.0	0.0	0.0	0.0
9.Automatic Program execution	0.9	0.0	0.9	0.9	0.9	0.9	0.9
10.Evolution of equivalent mutant	0.6	0.0	0.0	0.30	0.6	0.6	0.6
11.Multiple Language Support i.e.(Aspect-J/Ceaser-J etc.)	0.3	0.0	0.0	0.0	0.0	0.0	0.0
Overall Score		**0.70**	**1.65**	**2.30**	**2.60**	**2.80**	**5.05**

Delamaro and Maldonado[18]**Horgan and Mathur[16]Vincenzi *et al.* [20]**

Interpretation on Automation Tool: Proteum/AJ reported maximum score i.e. 5.05 out of 11 requirements with their ranking considered for evaluation. This shows that this tool is having higher confidence as compare to other testing tools because it reported high score. Maximum of requirements are full filled by Proteum/AJ mutation testing tools proved the strength of the tools among other considered tools for analysis. As stated in the beginning of this paper, it can be observed that all these six tools mentioned above in Table III, support the most of requirements for mutation testing in Aspect Oriented Programs. Apart from deriving test requirements according to the automated criteria, they all support automatic test execution. We can also notice that all tools target AspectJ programs. Mutants are generated at the source code level and all these tools support unit testing test phase.

From Table III, we can observe how tools for mutation testing of AO programs address the listed requirements. Anbalagan *et al.* [4] tool is limited to the creation and classification of mutants based on a very small set of mutation operators. No support for test case and mutant handling is provided. AjMutator, on the other hand provides better support than Advice Tracer, Cistron, Anbalagan and Xie's tool.

However it still misses some basic functionality such as mutation operator selection and proper mutant execution and analysis support (e.g. individual mutant execution and manual classification of mutants). Contrasting Proteum/AJ with the other previous tools, authors of the tool highlight that it improves test case handling features (e.g. importing and executing test cases into the running test application), enables mutant handling (e.g. individual mutant execution) and supports testing strategies (e.g. incremental selection of mutation operators and target aspects). Proteum/AJ allows the tester to manage mutants in different ways. For example, mutants can be created, recreated and individually selected for final execution. The execution can also be restricted to live mutants only, and these can be manually set as equivalent and vice versa, that is, equivalent mutants can be reset as alive. The tool also enables the tester to import and execute new test cases within an existing test project which is not supported by other tools. The mutation score can be computed at any time after the first tests have been executed. The size of the application under test is not limited by Proteum/AJ, whereas most of the tools restrict the program size. There are only minor dependencies between the test execution configuration and the tool. Proteum/AJ tool also produces mutant analysis reports that show the current mutation score and the mutated parts of the code for each mutant considered for testing.

Copyright © 2014 MECS

V. Conclusion And Future Scope

As per analysis, it can be concluded that the Proteum/AJ is more efficient tools, as compared to other tools for AspectJ restricted to above mentioned requirements or conditions. Proteum/AJ achieved the highest numeric weighted factor in analysis of mutation tool based on the most common requirements which are identified by various researchers for mutation testing in general. Purpose of the present work is not criticize any work because each one having some strengths and its limitation but in limited domain and with selected criteria.

Every tool supports its own technique and there is lack of common interface which makes it difficult to handle tool interface. There is a strong requirement of a tool, which work on one standard technique and full-fill all the essential requirements of mutation testing. Few of these tools discussed above require some additional tools support which should be eliminated and that feature should be integrated in the tool itself. AspectJ based system level mutation testing is not carried out, which need to be added. Performance of tool is major parameters while applying any testing technique either by considering the sequence applications or object oriented application by testing tools. Performance is not even considered in automated mutation testing tools we have analyzed.

Acknowledgment

The authors wish to acknowledge, Gautam Buddha University, Greater Noida and Amity University Uttar Pradesh, India. Both organizations provide research environment, and their faculties and scholars provide valuable suggestion during this analysis.

References

[1] Tassey G., "The economic impacts of inadequate infrastructure for software testing", *Technical Planning Report 02-3*, National Institute of Standards and Technology, Program Office Strategic Planning and Economic Analysis Group, May 2002.

[2] Zuhoor A. K., Woodward M., and Ramadhan H.A., "Critical analysis of the pie testability technique", *Software Quality Control*, 10(4):331-354, 2002.

[3] Filman R., Elrad T., Clarke S., "Aspect Oriented Software Development", *Addison-Wesley Publishing Company*, 2004.

[4] Anbalagan P. and Tao X., "Automated Generation of Pointcut Mutants for Testing Pointcuts in AspectJ Programs", in *Proceedings of the 19th International Symposium on Software Reliability Engineering (ISSRE'08),* IEEE Computer Society, pp. 239–248, 11-14 November 2008.

[5] Zhao C. and Alexander R.T.,"Testing AspectJ Programs using Fault-Based Testing", *Workshop on Testing Aspect Oriented Programs (WTAOP'07),* Vancouver, British Columbia, Canada, ACM, 2007.

[6] Alexander R.T., Bieman J.M. and Andrews A. A., "Towards the Systematic Testing of Aspect-Oriented Programs", *Technical Report CS-4-105, Colorado State University*, 2004.

[7] DeMillo R.A., Lipton R.J., Sayward F.G., "Hints on test data selection: help for the practicing programmer", *IEEE Computer, 11 (4), pp.* 34–41, 1978.

[8] Offutt A. J. and Untch R. H., "Mutation 2000: Uniting the Orthogonal", In Mutation 2000: Mutation Testing in the Twentieth and the Twenty First Centuries, pp. 45-55, 2000.

[9] DeMillo R. A. and Offutt A. J., "Constraint-based automatic test data generation", *IEEE Trans. Softw. Eng.*, 17(9), pp. 900– 910, 1991.

[10] Offutt A.J., "A Practical System for Mutation Testing: Help for the Common Programmer", *Twelfth International Conference on Testing Computer Software*, pp. 99-109, 1995.

[11] Andrews J.H., Briand L.C., Labiche Y., "Is mutation an appropriate tool for testing experiments?", In Proc. of 27th International Conference on Software Engineering (ICSE 2005), pp. 402–411, 2005.

[12] Anbalagan P. and Tao X., "Efficient Mutant Generation for Mutation Testing of Pointcuts in Aspect-Oriented Programs", *Mutation 2006 - ISSRE Workshops, 2006*.

[13] Lemos O. A. L., Ferrari F. C., Lopes C. V. and Maseiro P. C., "Testing Aspect Oriented Programming Pointcut Descriptors", *In Proceedings of the 2nd workshop on Testing aspect-oriented programs(WTAOP '06)*, Portland, ACM, 2006.

[14] Ferrari F.C., Maldonado J.C., Rashid A., "Mutation Testing for Aspect-Oriented Programs", *International Conference on Software Testing, Verification, and Validation*, pp. 52-61, IEEE 2008.

[15] Mortensen M. and Alexander R. T., "An approach for adequate testing of AspectJ programs", *In Proc. 1st Workshop on Testing Aspect-Oriented Programs*, 2005.

[16] Yamazaki Y., Sakurai K., Matsuura S., Masuhara H., Hashiura H., and Komiya S., "A unit testing framework for aspects without weaving", *In Proc. 1st Workshop on Testing Aspect-Oriented Programs*, 2005.

[17] AspectJ benchmark, 2004, available at http://www.sable.mcgill.ca/benchmarks/.

[18] Singh M., Mishra S., "Mutant generation for Aspect oriented Programs", *International Journal of Computer Science and Engineering*, Vol.1, pages 409-415, 2010.

[19] Delamare R., Baudry B. , Ghosh S., and Le T.Y., "A Test-Driven Approach to Developing Pointcut Descriptors in AspectJ", *in Proceedings of the 2nd International Conference on Software Testing Verification and Validation (ICST'09)*, pp. 376–38, 2009.

[20] Delamare R., Advicetracer. Available at: www.irisa.fr/triskell/Softwares/protos/advicetracer.

[21] Dahm M. and Zyl J. Van, Byte Code Engineering Library, April 2003. Available at: http://jakarta.apache.org/bcel/.

[22] Sun Microsystems. Java Reflection API. Online manual, 2001.

[23] Eclipse, AspectJ compiler 1.5, May 2005. Available at: http://eclipse.org/aspectj/.

[24] Arnold K., Gosling J., and Holmes D., "The Java Programming Language", *Addison-Wesley Longman*, 2000.

[25] Singh M, Gupta P.K., Mishra S., "Automated Test Data Generation for Mutation Using AspectJ Programs", *In Proceedings of ICIIP-2011*, IEEE, 2011.

[26] Jackson A. and Clarke S., "MuAspectJ: Mutant Generation to Support Measuring the Testability of AspectJ Programs", *Technical report, ACM*, 2009.

[27] Delamare R., Baudry B., and Le T. Y., "AjMutator: A Tool for The Mutation Analysis of AspectJ Pointcut

Descriptors", *in Proceedings of the 4th International Workshop on Mutation Analysis (MUTATION'09)*, pp. 200–204, 2009.
[28] Ferrari F.C., Nakagawa E.Y., Maldonado J.C., Rashid A., "Proteum/AJ: a mutation system for AspectJ programs", *in Proceedings of AOSD-11*, ACM, 2010.
[29] Do H. and Rothermel H., "A controlled experiment assessing test case prioritization techniques via mutation faults", In Proc. of the 21st IEEE International Conference on Software Maintenance (ICSM 2005), pp. 411–420, 2005.
[30] Horgan J.R., Mathur A., "Assessing testing tools in research and education", *IEEE Software*, 9(3), pp.61–69, 1992.
[31] Madeyski L., Radyk N., "Judy – a mutation testing tool for Java", *Published in IET Softw.*, Vol. 4, Issue 1, pp. 32–42, 2010.
[32] Delamaro, M. E.; Maldonado, J. C., "Proteum: A tool for the assessment of test adequacy for C programs", *In: Conference on Performability in Computing Systems (PCS)*, USA, pp. 79-95, 1996.
[33] Singh M., Mishra S. and Mall R., "Accessing and Evaluating AspectJ based Mutation Testing Tools", *published in Iinternational Journal of Computer Application*, pp. 33-38, 2011.
[34] Vincenzi A. M. R., Simao, A. S., Delamaro, M. E.,Maldonado, J. C., "Muta-Pro: Towards the definition of a mutation testing process", *Journal of the Brazilian Computer Society*, Vol.12, No.2, pp. 49-61, 2006.
[35] Ceccato, M., Tonella, P., Ricca, F., "Is AOP code easier or harder to test than OOP code?" *In Proceedings of the First Workshop on Testing Aspect-Oriented Programs (WTAOP 2005)*. Chicago, Illinois, 2005.
[36] Scholivé, M., Beroulle, V., Robach, C., Flottes, M.L., Rouzeyre, B., "Mutation Sampling Technique for the Generation of Structural Test Data", *published in proceedings of Design, Automation and Test in Europe (DATE'05)*, 2005.
[37] Moore I., "Jester and Pester," http://jester.sourceforge.net/, 2001.
[38] Chevalley P. and The ́venod-Fosse P., "A Mutation Analysis Tool for Java Programs", International Journal of Software Tools for Technology Transfer, Vol. 5, No. 1, pp. 90-103, Nov. 2002.
[39] Ma Y.S., Offutt J., and Kwon Y.R., "MuJava: An Automated Class Mutation System," Software Testing, Verification, and Reliability, Vol. 15, No. 2, pp. 97-133, 2005.
[40] Ma Y.S., Offutt J., and Kwon Y.R., "MuJava: A Mutation System for Java," Proc. 28th International Conference on Software Engg., pp. 827-830, 2006.
[41] Offutt J., Ma Y.S., Kwon Y.R., "An Experimental Mutation System for Java", ACM SIGSOFT Software Eng. Notes, Vol. 29, No. 5, pp. 1-4, 2004.
[42] Bradbury J.S., Cordy J.R., and Dingel J., "ExMAn: A Generic and Customizable Framework for Experimental Mutation Analysis", Proc. Second Workshop Mutation Analysis, pp. 57-62, 2006.
[43] Kim S.W., Harrold M.J., and Kwon Y.R., "MUGAMMA: Mutation Analysis of Deployed Software to Increase Confidence Assist Evolution", Proc. Second Workshop Mutation Analysis, pp. 10, 2006.
[44] Smith B.H., Williams L., "An Empirical Evaluation of the MuJava Mutation Operators", Proc. Third Workshop Mutation Analysis, pp. 193-202, Sept. 2007.
[45] Source Forge, "Jumble," http://jumble.sourceforge.net/, 2007.
[46] Polo M., Tendero S., and Piattini M., "Integrating Techniques and Tools for Testing Automation", Software Testing, Verification, and Reliability, Vol. 17, No. 1, pp. 3-39, 2007.
[47] Grun B.J., Schuler D., Zeller A., "The Impact of Equivalent Mutants", Proc. Fourth International Workshop Mutation Analysis, pp. 192-199, Apr. 2009.
[48] Tanaka A., "Equivalence Testing for Fortran Mutation System Using Data Flow Analysis", Master's Thesis, Georgia Inst. of Technology, 1981.
[49] Just R., Schweiggert F., Kapfhammer G.M., "MAJOR: An efficient and extensible tool for mutation analysis in a Java compiler", published with Proceedings 26th IEEE/ACM International Conference on Automated Software Engineering, pp. 612-615, 2011.
[50] Offutt A.J. and Untch R.H., "Mutation 2000: Uniting the Orthogonal", Proc. First Workshop Mutation Analysis, pp. 34-44, 2000.
[51] Hamlet R.G., "Testing programs with the aid of a compiler", IEEE Trans. on Soft. Engg., pp. 279-290, Vol. (4), Jul. 1977.
[52] Singh, P.K, Sangwan O.P. and Sharma A., "A Systematic Review on Fault Based Mutation Testing Techniques and Tools for Aspect-J Programs", published in proceedings of 3rd IEEE International Advance Computing Conference, IACC-2013, India, 22-23 Feb. 2013.

Mr. Pradeep Kumar Singh is an Assistant Professor in Computer Engineering at the Amity School of Engineering and Technology, Amity University, Uttar Pradesh, India. He is member of ACM, CSI and many professional bodies. He has published 10 papers in International Conferences and Journals of repute.

Dr. Om Prakash Sangwan is working as Assistant Professor in Department of Computer Science and Engineering of Gautam Buddha University, Greater Noida. Uttar Pradesh, India. He is Senior Member of ACM, CSI, IEEE and many professional bodies. He has filled two Patents and published 40 papers in International Conferences and Journals of repute. His major area of Interest includes Software Engineering, Object Oriented Software Engineering, Aspect Oriented Software Engineering, Soft Computing.

Dr. Arun Sharma is working as Associate Professor in Indira Gandhi Delhi Technical University for Women (IGDTUW), New Delhi, India. He is Senior Member of CSI, IEEE and many professional bodies. He has published 50 papers in International Conferences and Journals of repute. His major area of Interest includes Software Engineering, Object Oriented and Component Based Software Engineering.

Copyright © 2014 MECS