Literature Survey on Mutation Testing

Ali Parsai

University of Antwerp
ali.parsai@student.uantwerpen.be

Abstract

Mutation testing is one of the leading methods of testing the test-suites. In this article we review the literature about mutation testing in order to provide a guide for a developer who wants to design a mutation testing framework. We explore the diverse nature of mutation operators as a main ingridient to any mutation testing framework, and also discuss the features of the tools developed in the past decade.

1. Introduction

Since the dawn of software engineering, one of the problems developers faced was to be able to make sure their software is free of bugs. A bug is defined as a fault in the system which results in unexpected behavior [74]. Therefore, to avoid any complications resulting from the existence of bugs in the software, it is in the interest of the developer to *test* their software. Automated test suites are designed to handle this task.

The demand for adequate testing requires a test suite to be of a certain level of quality; Otherwise, the ability of a test suite to catch bugs comes into question. There are a lot of legacy systems with inadequate test suites which need improvement to become of a level which can be useful for trustworthy regression testing. This creates the need for an idea to assess the quality of a test suite in a reliable, repeatable and falsifiable way. Mutation testing [15] provides a formal method to determine the quality of a test suite by injecting intentional bugs into a system and counting the number of caught bugs. There are several studies which show that mutation testing is successful in simulating the real-life bugs a test suite might catch [4][37]. Hence, there is widespread interest in its usage as an analysis method for scientific purposes.

Even though the idea of mutation testing looks simple, in reality, it is hard to implement, adapt and run. Since changing the software in a way that fabricates a real bug is dependent on the software in question, mutation testing frameworks are usually dependent on the syntax of the language used in the software. In addition to this, technologies used in developing the software (e.g. build system, test suite runner) are also a limiting factor for implementing a general mutation testing framework for a particular software. This means that developing a universal mutation testing framework demands an enormous amount of work to be done.

In this study, a brief overview of the state of the literature in the last decade was done in order to gather information regarding the available tools and methods used in their development. The goal of this study is to help the design and/or adaptation of a mutation testing framework on a particular system by providing information regarding the existing efforts available in the literature. Different aspects of a mutation testing framework is discussed and statistical data is used to find out which approach have had more proponents.

The rest of the paper is structured as follows. In section 2, the background of the subject is discussed. In section 3 the research questions are discussed. In section 4, the method to search for articles and filtering the results is discussed. In sections 5, an analysis of the results is done. In section 6, the statistics extracted from the literature is provided; and in section 7, similar studies are brought into attention.

2. Background

The idea of mutation testing was first mentioned in a class paper by Lipton [54] and later developed by DeMillo, Lipton and Sayward [15]. The first implementation of a mutation testing tool was done by Timothy Budd in 1980 [8]. However, the computationally demanding nature of mutation testing made it a difficult subject for empirical research and the progress in the subject was very slow. Advances in hardware capabilities and the increase in demand of the market due to availability of legacy code caused a resurgence in the topic and since late 1990's the topic has been a hot trend among software engineers.

Mutation testing is the process of injecting bugs into software and counting the number of caught bugs. This procedure is run in the following manner: First, faulty versions of the software is created by applying a single change in the system. This change is created by a mutation operator. A mutation operator is a piece of software which changes the system under test in such a way that it includes a single bug. After generating the *mutants*, the test suite is run on each of these mutants. If there is an error or failure during the execution of the test suite, the mutant is regarded as killed. However, if all tests pass, it means that the bug could not be caught by the test suite and the mutant has survived. If two mutants return the same output for all possible inputs, they are called equivalent mutants. The final result is the number of killed mutants excluding the number of equivalent mutants divided by the number of all generated mutants excluding the number of equivalent mutants.

The mutation operator is an important part of this procedure. The more mutation operators there is, more mutants will be generated and as a result the amount of computations needed is increased. Moreover, if the mutation operators are not sufficiently different, they increase the chance of creating equivalent mutants. Thus, there is an interest to find which subset of mutant operators would result in sufficient testing of the system. There has been multiple articles regarding this topic [53][52][36].

3. Research Goals

This study is designed towards those who need to acquire relevant knowledge about the current state of the art in the topic of mutation testing. This is useful for those who need to design a new mutation testing framework from scratch in order to analyze a system which does not work with the available tools, or those who are adapting an already existing framework into a usable system for their software based on their own criteria. This study also provides a simple guide for choosing a tool based on its capabilities in the form of an overview.

In order to achieve the goal mentioned above, the following research questions must be answered:

- **RQ1.** What mutation operators are essential in developing a mutation testing framework? Are these mutation operators applicable to every context, or are they specific to a special category of softwares?
- **RQ2.** Which tools implement these mutation operators? What programming languages are supported by each tool? What is the availability status of the tool?

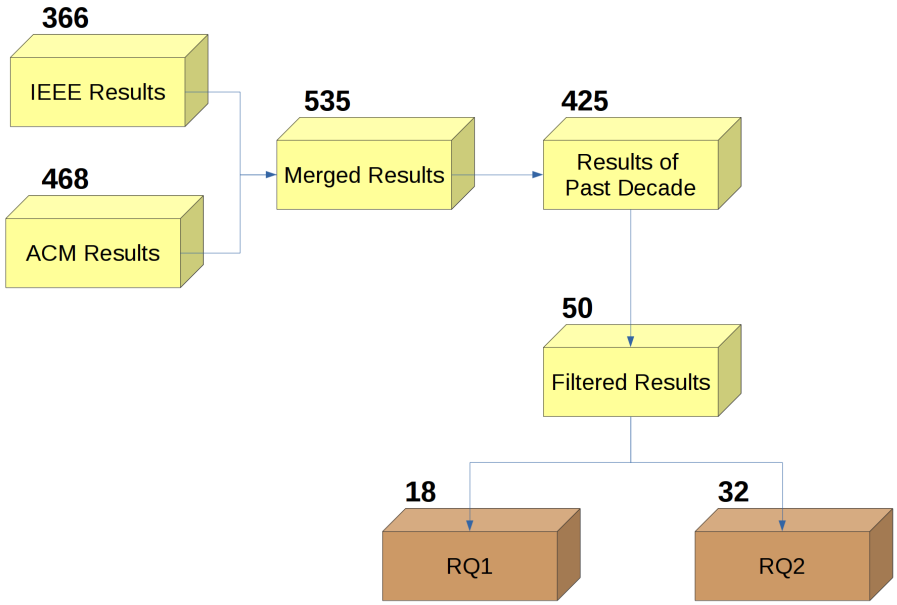

Figure 1: *Article selection procedure*

4. Survey Method

To conduct this survey, several automated tools were used to gather, merge and handle the results. In order to be able to use automated tools, the reference libraries were limited to IEEE[1] and ACM[2]. A customized version of SLRTool[3] was used in order to retrieve preliminary results from the aforementioned libraries. The search was conducted using two queries — "Mutation Testing" and "Mutation Analysis" — in order to find all relevant results.

Then, Mendeley[4] was used to merge all resulting files and to remove duplicate entries. After removing all the results from before year 2004, the resulting entries were manually inspected to be categorized according to the research questions. This lead to the final count of 50 results overall. Of this 50 articles, 18 were about the mutation operators and 32 were related to the developed tools. An overall summery of the survey method can be seen in Figure 1.

5. Analysis

After narrowing down the results of the survey and categorizing them according to the subject, the resulting articles were read to seek the answers of our research questions. In the following sections, I try to summarize the information acquired through this survey.

5.1. Essential Mutation Operators

The first set of the mutation operators designed were reported in [42]. These operators which work on very basic entities were designed for the tool Mothra which was designed to mutate

[1] http://ieeexplore.ieee.org
[2] http://dl.acm.org
[3] http://github.com/javipeg/SLRtool
[4] http://www.mendeley.com

Mutation Operator	Description
AAR	array reference for array reference replacement
ABS	absolute value insertion
ACR	array reference for constant replacement
AOR	arithmetic operator replacement
ASR	array reference for scalar variable replacement
CAR	constant for array reference replacement
CNR	comparable array name replacement
CRP	constant replacement
CSR	constant for scalar variable replacement
DER	DO statement alterations
DSA	DATA statement alterations
GLR	GOTO label replacement
LCR	logical connector replacement
ROR	relational operator replacement
RSR	RETURN statement replacement
SAN	statement analysis
SAR	scalar variable for array reference replacement
SCR	scalar for constant replacement
SDL	statement deletion
SRC	source constant replacement
SVR	scalar variable replacement
UOI	unary operator insertion

Figure 2: *Mothra mutation operators (adapted from [33])*

FORTRAN77 programming language (Figure 2). In 1996, Offutt et al [53] determined that only a selection of few mutation operators are enough to produce the same coverage with a four-fold reduction of the number of mutants. This reduced set of operators remained more or less intact in the future research papers (Figure 3).

With the popularity of the object-oriented programming, in order to adapt mutation testing to the new concept, the need for new mutation operators was felt. Several studies proposed new mutation operators [47] [9], and some of them were designed to prove the usefulness of object-oriented operators [44] [17]. Ahmed et al in [2] did a complete survey on this subject. As a sample, object-oriented mutation operators which were developed into MuJava [48] can be seen in Figure 4.

During past decade, the focus of the researchers were on creating new mutation operators for special purposes such as targeting certain security problems [66] [75] or language specific mutation operators [1] [7] [67]. These mutation operators, even though important in their own context, does not relegate into the general concept of mutation testing. Therefore, one needs to study them in case of designing a system for a specific purpose, but mostly they are ignored in general. As seen in section 6, most popular mutation operators used in the tools available are by far the classical ones.

5.2. Mutation Analysis Tools

There are several tools developed in the past decade for the purpose of mutation analysis. An overview of the specifications of these tools can be seen in Figure 5. What follows is a small description for each of these tools.

Operator	Description
AOR	Arithmetic Operator Replacement
AOD	Arithmetic operator Deletion
AOI	Arithmetic operator Insertion
ROR	Relational Operator Replacement
COR	Conditional Operator Replacement
COD	Conditional operator Deletion
COI	Conditional operator Insertion
SOR	Shift Operator Replacement
LOR	Logical Operator Replacement
LOD	Logical operator Deletion
LOI	Logical operator Insertion
ASR	Assignment Operator Replacement

Figure 3: *Reduced-set mutation operators (adapted from [49])*

Operators	Description
AMC	Access modifier change
IHD	Hiding variable deletion
IHI	Hiding variable insertion
IOD	Overriding method deletion
IOP	Overridden method calling position change
IOR	Overridden method rename
ISK	*super* keyword deletion
IPC	Explicit call of a parent's constructor deletion
PNC	*new* method call with child class type
PMD	Instance variable declaration with parent class type
PPD	Parameter variable declaration with child class type
PRV	Reference assignment with other compatible type
OMR	Overloading method contents change
OMD	Overloading method deletion
OAO	Argument order change
OAN	Argument number change
JTD	*this* keyword deletion
JSC	*static* modifier change
JID	Member variable initialization deletion
JDC	Java-supported default constructor create
EOA	Reference assignment and content assignment replacement
EOC	Reference comparison and content comparison replacement
EAM	Accessor method change
EMM	Modifier method change

Figure 4: *Object-oriented mutation operators (adapted from [47])*

MuJava MuJava is a publicly available mutation system for Java that supports both traditional statement-level mutants and newer inter-class mutants. Relevant articles: [56] [48] [49] [68] [58]

SQLMutation SQLMutation is a tool to automatically generate mutants of SQL database queries [72].

Certitude Certitude is a commercial software tool performing mutation analysis on C programs with the focus on the microelectronics industry [28].

ExMAn ExMAn is an automated, general and flexible mutation analysis framework which allows for the comparison of different quality assurance techniques such as testing, model checking, and static analysis on C and Java code [6].

MUGAMMA MUGAMMA implements a software system so that when it executes in the field, it will determine whether users' executions would have killed mutants (without actually executing the mutants) [40].

MuClipse MuClipse is an automated mutation testing plug-in for Eclipse [68].

CSAW CSAW is a lightweight C language mutation tool [21].

Jumble Jumble is a byte code level mutation testing tool for Java which inter-operates with JUnit [30].

MUFORMAT MUFORMAT is a prototype tool which is targeted at format string bugs in C [66].

CREAM CREAM is a mutant generator for C# [18].

MUSIC MUSIC is MUtation-based SQL Injection vulnerabilities Checking (testing) tool that automatically generates mutants for the applications written in Java Server Pages (JSP) and performs mutation analysis [65].

MILU Milu is an efficient and flexible C mutation testing tool designed for both first order and higher order mutation testing [34].

Javalanche Javalanche is an open source framework for mutation testing Java programs with a special focus on automation, efficiency, and effectiveness which assesses the impact of individual mutations to effectively weed out equivalent mutants [63].

GAmera GAmera is an automatic mutant generator for WS-BPEL compositions. It is composed by three different elements: an analyzer, a mutant generator and a system that executes and evaluates the mutants. GAmera is based in genetic algorithms and attempts to minimize the number of generated mutants, independently of the number and type of mutation operators, without losing relevant information. It can also detect potentially equivalent mutants allowing to improve the quality of the test suite [20] [19].

Name	Year	Language	Operators	Availability
MuJava	2004	Java	Classical, Object-Oriented	Free
SQLMutation	2006	SQL	SQL Specific	Free
Certitude	2006	C, C++	Classical	Commercial
ExMAn	2006	C, Java	Classical	Free
MUGAMMA	2006	Java	Classical	Free
MuClipse	2007	Java	Classical, Object-Oriented	Free
CSAW	2007	C	Classical	Free
Jumble	2007	Java	Classical, Object-Oriented	Free
MUFORMAT	2008	C	FSB	Not Available
CREAM	2008	C#	Classical	Not Available
MUSIC	2008	SQL	SQL Specific	Not Available
MILU	2008	C	Classical, Higher-Order	Free
Javalanche	2009	Java	Classical	Free
GAmera	2009	WS-BPEL	Classical, WS-BPEL Specific	Free
AjMutator	2009	AspectJ	AspectJ Specific	Free
Proteum/AJ	2010	AspectJ	AspectJ Specific	Not Available
Judy	2010	Java	Classical, Object-Oriented	Free
X-Mut	2010	XSLT	Classical	Not Available
MutMut	2010	Java	Classical	Not Available
SMutant	2011	SmallTalk	Classical	Free
MAJOR	2011	Java	Classical	Free
ConSMutate	2012	SQL	SQL Specific	Not Available
Bacterio	2012	Java	Classical	Not Available
SMT-C	2012	C, C++	Classical, Semantical	Not Available
MuCheck	2014	Haskell	Haskell Specific	Free

Figure 5: *Mutation testing tools*

AjMutator AjMutator is a tool for the mutation analysis of PCDs[5]. AjMutator implements several mutation operators that introduce faults in the PCDs to generate a set of mutants. AjMutator classifies the mutants according to the set of join points they match compared to the set of join points matched by the initial PCD [13].

Proteum/AJ Proteum/AJ realizes a set of requirements for mutation-based testing tools and overcomes some limitations identified in previous tools for aspect-oriented programs [24] [23].

Judy Judy presents an innovative approach to mutation testing that takes advantage of a novel aspect-oriented programming mechanism, called 'pointcut and advice', to avoid multiple compilation of mutants and to speed up mutation testing [50].

X-Mut X-Mut is a tool which provides an adaptation of mutation analysis for XSLT language [45].

MutMut MutMut is a tool for Mutation Testing of Multithreaded Code [26].

SMutant SMutant is the first mutation testing tool for Smalltalk programs [25].

MAJOR MAJOR is a fault seeding and mutation analysis tool that is integrated into the Java Standard Edition compiler as a non-invasive enhancement for use in any Java-based development environment [39] [37].

ConSMutate ConSMutate is a mutation testing framework which guides concolic testing using mutation analysis for test case generation for database applications [60] [59].

Bacterio Bacterio is a Java mutation testing tool that automates the tasks to perform mutation analyses and that implements a set of mutation techniques that reduce the costs of mutation and the execution mutant time drastically [73].

SMT-C SMT-C is a semantic mutation testing tool for C [11].

MuCheck MuCheck is a mutation testing tool for Haskell programs [43].

6. Statistics

A quick look into the available data reveals that the subject has kept its importance throughout the past decade steadily. Figure 6 shows the number of articles and number of tools developed in each year of the past decade. As it can be seen, the trend has remained more or less steady for the past decade with the average of 3.2 articles per year, and 2.5 tools per year. An interesting fact that can be observed in the Figure 6 is that during 2013, no tools have been developed, and only one tool was developed in 2014. However, 3 articles were written during this period about the previously available tools which shows that the tools still can be expanded by incorporating new ideas.

Out of the 25 tools developed in the past decade, 10 were developed for Java, and 7 for the C family (C, C++, C#). Interestingly, SQL holds the third place with 3 tools developed for it which shows the interest due to the rise in the demand of database applications. Figure 7 shows all the languages the tools have been developed for and their distribution.

[5]Pointcut Descriptors

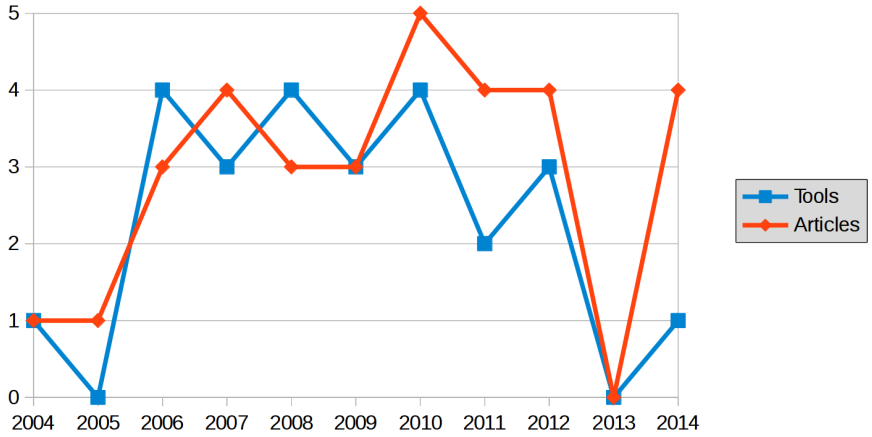

Figure 6: *Subject trend*

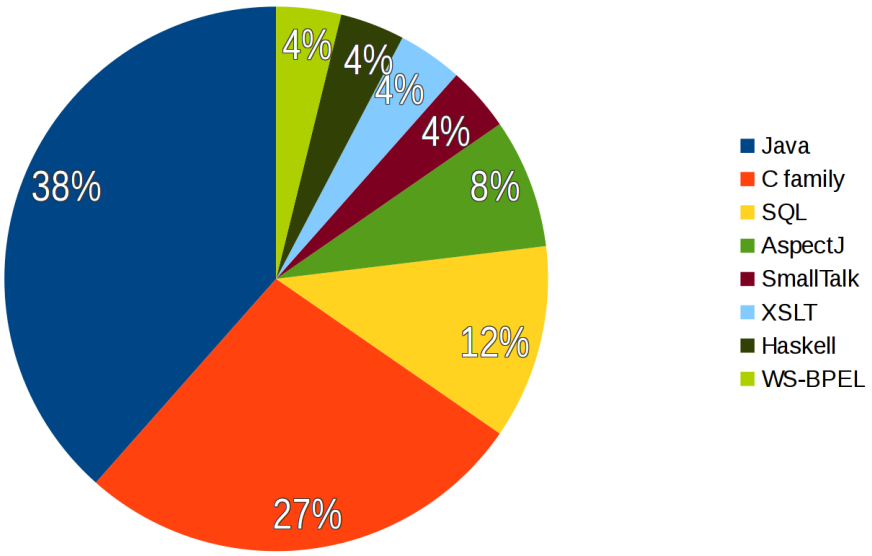

Figure 7: *Target languages of the tools*

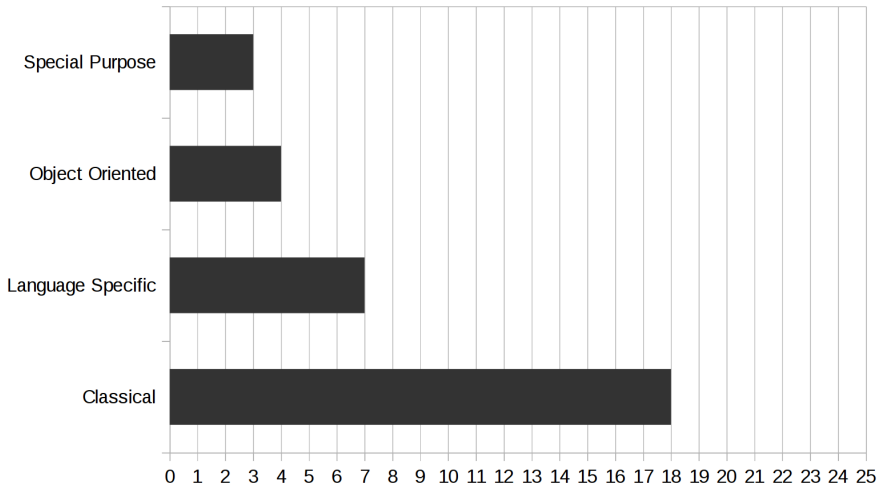

Figure 8: *Mutation operators used in the tools*

One interesting phenomena observed from this data is the fact that most of the tools use classical mutation operators and only a few of them use object-oriented mutation operators. This, combined with the fact that the majority of the tools target Object Oriented languages show that there is a lack of interest in developing object-oriented mutation operators. This may be due to the fact that the empirical research proving the usefulness of such operators is still incomplete; However the existing empirical research supports this claim [44]. Figure 8 shows the usage of the mutation operators in tools.

7. Related Work

There are other surveys tackling the subject of mutation testing. One of the surveys which does a comprehensive analysis of the subject is Jia et al 2011 [33]. In this survey, the process of mutation testing is explained, and its problems and solutions in the literature are discussed. Also, empirical analysis of a lot of the tools discussed in this report is provided. Offutt et al in [55] discuss the history of mutation testing and the state of the art now, and provides insight into the future of the field. A good reference for analysis of the mutation testing tools for java is Delahaye et al 2013 [12]. In the mentioned article, mutation testing tools for java are compared based on efficiency, compatibility with current technologies and multiple other factors.

References

[1] Robin Abraham and Martin Erwig. Mutation Operators for Spreadsheets. *IEEE Trans. Softw. Eng.*, 35(1):94–108, January 2009.

[2] Z Ahmed, M Zahoor, and I Younas. Mutation operators for object-oriented systems: A survey, 2010.

[3] R.T. Alexander, J.M. Bieman, S. Ghosh, and Bixia Ji. Mutation of java objects. In *Software Reliability Engineering, 2002. ISSRE 2003. Proceedings. 13th International Symposium on*, pages 341–351, 2002.

[4] J.H. Andrews, L.C. Briand, and Y. Labiche. Is mutation an appropriate tool for testing experiments? [software testing]. In *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*, pages 402–411, May 2005.

[5] Juan Boubeta-Puig, Inmaculada Medina-Bulo, A A-DomiÌĄ nguez, and Antonio García-Domínguez. Analogies and Differences Between Mutation Operators for WS-BPEL 2.0 and Other Languages. In *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on*, ICSTW '11, pages 398–407, Washington, DC, USA, 2011. Dept. of Comput. Languages & Syst., Univ. of Cadiz, Cadiz, Spain, IEEE Computer Society.

[6] Jeremy S Bradbury, James R Cordy, and Juergen Dingel. ExMAn: A Generic and Customizable Framework for Experimental Mutation Analysis. In *Proceedings of the Second Workshop on Mutation Analysis*, MUTATION '06, page 4, Washington, DC, USA, 2006. Sch. of Comput., Queen"s Univ., Kingston, ON, IEEE Computer Society.

[7] Jeremy S Bradbury, James R Cordy, and Juergen Dingel. Mutation Operators for Concurrent Java (J2SE 5.0). In *Proceedings of the Second Workshop on Mutation Analysis*, MUTATION '06, pages 11—-, Washington, DC, USA, 2006. Sch. of Comput., Queen"s Univ., Kingston, ON, IEEE Computer Society.

[8] Timothy Alan Budd. *Mutation Analysis of Program Test Data*. PhD thesis, New Haven, CT, USA, 1980. AAI8025191.

[9] Huo Yan Chen and Su Hu. Two New Kinds of Class Level Mutants for Object-Oriented Programs, 2006.

[10] John A Clark, Haitao Dan, and Robert M Hierons. Semantic Mutation Testing. In *Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops*, ICSTW '10, pages 100–109, Washington, DC, USA, 2010. IEEE Computer Society.

[11] Haitao Dan and Robert M Hierons. SMT-C: A Semantic Mutation Testing Tools for C. In *Proceedings of the 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, ICST '12, pages 654–663, Washington, DC, USA, 2012. Sch. of Inf. Syst., Comput. & Math., Brunel Univ., Uxbridge, UK, IEEE Computer Society.

[12] M. Delahaye and L." du Bousquet. A comparison of mutation analysis tools for java. In *Quality Software (QSIC), 2013 13th International Conference on*, pages 187–195, July 2013.

[13] Romain Delamare, Benoit Baudry, and Yves Le Traon. AjMutator: A Tool for the Mutation Analysis of AspectJ Pointcut Descriptors. In *Proceedings of the IEEE International Conference on Software Testing, Verification, and Validation Workshops*, ICSTW '09, pages 200–204, Washington, DC, USA, 2009. IEEE Computer Society.

[14] Marcio Eduardo Delamaro, Jeff Offutt, and Paul Ammann. Designing Deletion Mutation Operators. In *Proceedings of the 2014 IEEE International Conference on Software Testing, Verification, and Validation*, ICST '14, pages 11–20, Washington, DC, USA, 2014. IEEE Computer Society.

[15] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on test data selection: Help for the practicing programmer. *Computer*, 11(4):34–41, April 1978.

[16] Lin Deng, Jeff Offutt, and Nan Li. Empirical Evaluation of the Statement Deletion Mutation Operator. In *Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on*, ICST '13, pages 84–93, Washington, DC, USA, 2013. Software Eng., George Mason Univ., Fairfax, VA, USA, IEEE Computer Society.

[17] Anna Derezinska and Marcin Rudnik. Quality Evaluation of Object-oriented and Standard Mutation Operators Applied to C# Programs. In *Proceedings of the 50th International Conference on Objects, Models, Components, Patterns*, TOOLS'12, pages 42–57, Berlin, Heidelberg, 2012. Springer-Verlag.

[18] Anna Derezinska and Anna Szustek. Tool-Supported Advanced Mutation Approach for Verification of C# Programs. In *Dependability of Computer Systems, 2008. DepCos-RELCOMEX '08. Third International Conference on*, DEPCOS-RELCOMEX '08, pages 261–268, Washington, DC, USA, 2008. Inst. of Comput. Sci., Warsaw Univ. of Technol., Warsaw, IEEE Computer Society.

[19] Juan-José Domínguez-Jiménez, Antonia Estero-Botaro, Antonio García-Domínguez, and Inmaculada Medina-Bulo. GAmera: A Tool for WS-BPEL Composition Testing Using Mutation Analysis. In *Proceedings of the 10th International Conference on Web Engineering*, ICWE'10, pages 490–493, Berlin, Heidelberg, 2010. Springer-Verlag.

[20] Juan José Domínguez-Jiménez, Antonia Estero-Botaro, Antonio García-Domínguez, Inmaculada Medina-Bulo, J J Dominguez-Jimenez, Antonia Estero-Botaro, A Garcia-Dominguez, and Inmaculada Medina-Bulo. GAmera: An Automatic Mutant Generation System for WS-BPEL Compositions. In *Web Services, 2009. ECOWS '09. Seventh IEEE European Conference on*, ECOWS '09, pages 97–106, Washington, DC, USA, 2009. Dept. Comput. Languages & Syst., Univ. of Cadiz, Cadiz, Spain, IEEE Computer Society.

[21] M. Ellims, D. Ince, and Marian Petre. The csaw c mutation tool: Initial results. In *Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION, 2007. TAICPART-MUTATION 2007*, pages 185–192, Sept 2007.

[22] Fabiano Cutigi Ferrari, José Carlos Maldonado, and Awais Rashid. Mutation Testing for Aspect-Oriented Programs. In *Software Testing, Verification, and Validation, 2008 1st International Conference on*, ICST '08, pages 52–61, Washington, DC, USA, 2008. Dept. of Comput. Syst., Sao Paulo Univ., Sao Carlos, IEEE Computer Society.

[23] Fabiano Cutigi Ferrari, Elisa Yumi Nakagawa, José Carlos Maldonado, and Awais Rashid. Proteum/AJ: A Mutation System for AspectJ Programs. In *Proceedings of the Tenth International Conference on Aspect-oriented Software Development Companion*, AOSD '11, pages 73–74, New York, NY, USA, 2011. ACM.

[24] Fabiano Cutigi Ferrari, Elisa Yumi Nakagawa, Awais Rashid, and José Carlos Maldonado. Automating the Mutation Testing of Aspect-oriented Java Programs. In *Proceedings of the 5th Workshop on Automation of Software Test*, AST '10, pages 51–58, New York, NY, USA, 2010. ACM.

[25] Milos Gligoric, Sandro Badame, and Ralph Johnson. SMutant: A Tool for Type-sensitive Mutation Testing in a Dynamic Language. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, pages 424–427, New York, NY, USA, 2011. ACM.

[26] Milos Gligoric, Vilas Jagannath, and Darko Marinov. MuTMuT: Efficient Exploration for Mutation Testing of Multithreaded Code. In *Software Testing, Verification and Validation (ICST), 2010 Third International Conference on*, ICST '10, pages 55–64, Washington, DC, USA, 2010. Dept. of Comput. Sci., Univ. of Illinois at Urbana-Champaign, Urbana, IL, USA, IEEE Computer Society.

[27] B J M Grun, D Schuler, and A Zeller. The Impact of Equivalent Mutants, 2009.

[28] Mark Hampton and Stephane Petithomme. Leveraging a Commercial Mutation Analysis Tool For Research. In *Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION*, TAICPART-MUTATION '07, pages 203–209, Washington, DC, USA, 2007. IEEE Computer Society.

[29] Siamak Haschemi and Stephan WeiÃ§leder. A generic approach to run mutation analysis. In Leonardo Bottaci and Gordon Fraser, editors, *Testing - Practice and Research Techniques*, volume 6303 of *Lecture Notes in Computer Science*, pages 155–164. Springer Berlin Heidelberg, 2010.

[30] S A Irvine, T Pavlinic, L Trigg, J G Cleary, S Inglis, and M Utting. Jumble Java Byte Code to Measure the Effectiveness of Unit Tests, 2007.

[31] Changbing Ji, Zhenyu Chen, Baowen Xu, and Ziyuan Wang. A New Mutation Analysis Method for Testing Java Exception Handling. In *Proceedings of the 2009 33rd Annual IEEE International Computer Software and Applications Conference - Volume 02*, COMPSAC '09, pages 556–561, Washington, DC, USA, 2009. IEEE Computer Society.

[32] Yue Jia and M Harman. Constructing Subtle Faults Using Higher Order Mutation Testing, 2008.

[33] Yue Jia and M. Harman. An analysis and survey of the development of mutation testing. *Software Engineering, IEEE Transactions on*, 37(5):649–678, Sept 2011.

[34] Yue Jia and Mark Harman. MILU: A Customizable, Runtime-Optimized Higher Order Mutation Testing Tool for the Full C Language. In *Proceedings of the Testing: Academic & Industrial Conference - Practice and Research Techniques*, TAIC-PART '08, pages 94–98, Washington, DC, USA, 2008. King"s Coll. London, London, IEEE Computer Society.

[35] Ying Jiang, Shan-Shan Hou, Jin-Hui Shan, Lu Zhang, and Bing Xie. Contract-Based Mutation for Testing Components. In *Proceedings of the 21st IEEE International Conference on Software Maintenance*, ICSM '05, pages 483–492, Washington, DC, USA, 2005. IEEE Computer Society.

[36] R Just, G M Kapfhammer, and F Schweiggert. Using Non-redundant Mutation Operators and Test Suite Prioritization to Achieve Efficient and Scalable Mutation Analysis, 2012.

[37] René Just. The Major Mutation Framework: Efficient and Scalable Mutation Analysis for Java. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ISSTA 2014, pages 433–436, New York, NY, USA, 2014. ACM.

[38] René Just, Gregory M. Kapfhammer, and Franz Schweiggert. Using conditional mutation to increase the efficiency of mutation analysis. In *Proceedings of the 6th International Workshop on Automation of Software Test*, AST '11, pages 50–56, New York, NY, USA, 2011. ACM.

[39] Rene Just, Franz Schweiggert, and Gregory M Kapfhammer. MAJOR: An efficient and extensible tool for mutation analysis in a Java compiler. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, ASE '11, pages 612–615, Washington, DC, USA, 2011. Dept. of Appl. Inf. Process., Ulm Univ., Ulm, Germany, IEEE Computer Society.

[40] Sang-Woon Kim, M.J. Harrold, and Yong-Rae Kwon. Mugamma: Mutation analysis of deployed software to increase confidence and assist evolution. In *Mutation Analysis, 2006. Second Workshop on*, pages 10–10, Nov 2006.

[41] Sunwoo Kim, John A. Clark, and John A. McDermid. Class mutation: Mutation testing for object-oriented programs. In *PROC. NET.OBJECTDAYS*, pages 9–12, 2000.

[42] K. N. King and A. Jefferson Offutt. A fortran language system for mutation-based software testing. *Software: Practice and Experience*, 21(7):685–718, 1991.

[43] Duc Le, Mohammad Amin Alipour, Rahul Gopinath, and Alex Groce. MuCheck: An Extensible Tool for Mutation Testing of Haskell Programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ISSTA 2014, pages 429–432, New York, NY, USA, 2014. ACM.

[44] Hyo-Jeong Lee, Yu-Seong Ma, and Yong-Rae Kwon. Empirical Evaluation of Orthogonality of Class Mutation Operators. In *Proceedings of the 11th Asia-Pacific Software Engineering Conference*, APSEC '04, pages 512–518, Washington, DC, USA, 2004. IEEE Computer Society.

[45] Francesca Lonetti and Eda Marchetti. X-MuT: A Tool for the Generation of XSLT Mutants. In *Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the*, QUATIC '10, pages 280–285, Washington, DC, USA, 2010. Consiglio Naz. delle Ric., Ist. di Scienza e Tecnol. dell"Inf. A. Faedo, Pisa, Italy, IEEE Computer Society.

[46] Yu-Seung Ma, Mary Jean Harrold, and Yong-Rae Kwon. Evaluation of mutation testing for object-oriented programs. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 869–872, New York, NY, USA, 2006. ACM.

[47] Yu-Seung Ma, Yong-Rae Kwon, and J. Offutt. Inter-class mutation operators for java. In *Software Reliability Engineering, 2002. ISSRE 2003. Proceedings. 13th International Symposium on*, pages 352–363, 2002.

[48] Yu-Seung Ma, Jeff Offutt, and Yong Rae Kwon. MuJava: An Automated Class Mutation System: Research Articles. *Softw. Test. Verif. Reliab.*, 15(2):97–133, June 2005.

[49] Yu-Seung Ma, Jeff Offutt, and Yong-Rae Kwon. MuJava: A Mutation System for Java. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 827–830, New York, NY, USA, 2006. ACM.

[50] L Madeyski and N Radyk. Judy - a mutation testing tool for java, 2010.

[51] Jean-Marie Mottu, Benoit Baudry, and Yves Le Traon. Mutation analysis testing for model transformations. In Arend Rensink and Jos Warmer, editors, *Model Driven Architecture ÃćÂĂÂŞ Foundations and Applications*, volume 4066 of *Lecture Notes in Computer Science*, pages 376–390. Springer Berlin Heidelberg, 2006.

[52] AS. Namin, J.H. Andrews, and D. Murdoch. Sufficient mutation operators for measuring test effectiveness. In *Software Engineering, 2008. ICSE '08. ACM/IEEE 30th International Conference on*, pages 351–360, May 2008.

[53] A. Jefferson Offutt, Ammei Lee, Gregg Rothermel, Roland H. Untch, and Christian Zapf. An experimental determination of sufficient mutant operators. *ACM Trans. Softw. Eng. Methodol.*, 5(2):99–118, April 1996.

[54] A. Jefferson Offutt and Ronald H. Untch. Mutation testing for the new century. chapter Mutation 2000: Uniting the Orthogonal, pages 34–44. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[55] Jeff Offutt. A mutation carol: Past, present and future. *Information and Software Technology*, 53(10):1098 – 1107, 2011. Special Section on Mutation Testing.

[56] Jeff Offutt, Yu-Seung Ma, and Yong-Rae Kwon. An Experimental Mutation System for Java. *SIGSOFT Softw. Eng. Notes*, 29(5):1–4, September 2004.

[57] Elmahdi Omar, Sudipto Ghosh, and Darrell Whitley. HOMAJ: A Tool for Higher Order Mutation Testing in AspectJ and Java. In *Proceedings of the 2014 IEEE International Conference on Software Testing, Verification, and Validation Workshops*, ICSTW '14, pages 165–170, Washington, DC, USA, 2014. IEEE Computer Society.

[58] P Pecev, B Markoski, L Ratgeber, D Lacmanovic, and Z Ivankovic. Making muJava more accessible, 2012.

[59] Tanmoy Sarkar, Samik Basu, and Johnny Wong. iConSMutate: Concolic Testing of Database Applications Using Existing Database States Guided by SQL Mutants. In *Proceedings of the 2014 11th International Conference on Information Technology: New Generations*, ITNG '14, pages 479–484, Washington, DC, USA, 2014. IEEE Computer Society.

[60] Tanmoy Sarkar, Samik Basu, and Johnny S Wong. ConSMutate: SQL Mutants for Guiding Concolic Testing of Database Applications. In *Proceedings of the 14th International Conference on Formal Engineering Methods: Formal Methods and Software Engineering*, ICFEM'12, pages 462–477, Berlin, Heidelberg, 2012. Springer-Verlag.

[61] Sharmila Savarimuthu and Michael Winikoff. Mutation Operators for Cognitive Agent Programs. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '13, pages 1137–1138, Richland, SC, 2013. International Foundation for Autonomous Agents and Multiagent Systems.

[62] David Schuler, Valentin Dallmeier, and Andreas Zeller. Efficient mutation testing by checking invariant violations. In *Proceedings of the Eighteenth International Symposium on Software Testing and Analysis*, ISSTA '09, pages 69–80, New York, NY, USA, 2009. ACM.

[63] David Schuler and Andreas Zeller. Javalanche: Efficient Mutation Testing for Java. In *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC/FSE '09, pages 297–298, New York, NY, USA, 2009. ACM.

[64] David Schuler and Andreas Zeller. (Un-)Covering Equivalent Mutants. In *Software Testing, Verification and Validation (ICST), 2010 Third International Conference on*, ICST '10, pages 45–54, Washington, DC, USA, 2010. Saarland Univ., Saarbrucken, Germany, IEEE Computer Society.

[65] Hossain Shahriar and Mohammad Zulkernine. MUSIC: Mutation-based SQL Injection Vulnerability Checking. In *Proceedings of the 2008 The Eighth International Conference on Quality Software*, QSIC '08, pages 77–86, Washington, DC, USA, 2008. IEEE Computer Society.

[66] Hossain Shahriar and Mohammad Zulkernine. Mutation-Based Testing of Format String Bugs. In *Proceedings of the 2008 11th IEEE High Assurance Systems Engineering Symposium*, HASE '08, pages 229–238, Washington, DC, USA, 2008. Sch. of Comput., Queen"s Univ., Kingston, ON, IEEE Computer Society.

[67] Rodolfo Adamshuk Silva, Simone do Rocio Senger de Souza, and Paulo Sergio Lopes de Souza. Mutation operators for concurrent programs in MPI. In *Test Workshop (LATW), 2012 13th Latin American*, LATW '12, pages 1–6, Washington, DC, USA, 2012. Instituto de Ciências Matemáticas e de Computação - ICMC, Universidade de São Paulo, USP, São Carlos, Brazil, IEEE Computer Society.

[68] Ben H Smith and Laurie Williams. An Empirical Evaluation of the MuJava Mutation Operators. In *Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION*, TAICPART-MUTATION '07, pages 193–202, Washington, DC, USA, 2007. North Carolina State Univ., Raleigh, IEEE Computer Society.

[69] Ben H. Smith and Laurie Williams. Should software testers use mutation analysis to augment a test set? *Journal of Systems and Software*, 82(11):1819 – 1832, 2009. SI: {TAIC} {PART} 2007 and {MUTATION} 2007.

[70] BenH. Smith and Laurie Williams. On guiding the augmentation of an automated test suite via mutation analysis. *Empirical Software Engineering*, 14(3):341–369, 2009.

[71] Massimo Tisi, FrÃČÂľdÃČÂľric Jouault, Piero Fraternali, Stefano Ceri, and Jean BÃČÂľzivin. On the use of higher-order model transformations. In RichardF. Paige, Alan Hartman, and Arend Rensink, editors, *Model Driven Architecture - Foundations and Applications*, volume 5562 of *Lecture Notes in Computer Science*, pages 18–33. Springer Berlin Heidelberg, 2009.

[72] Javier Tuya, Ma Jose Suarez-Cabal, and Claudio de la Riva. Sqlmutation: A tool to generate mutants of sql database queries. In *Proceedings of the Second Workshop on Mutation Analysis*, MUTATION '06, pages 1–, Washington, DC, USA, 2006. IEEE Computer Society.

[73] Macario Polo Usaola, Pedro Reales Mateo, and Macario Polo Usaola. Bacterio: Java mutation testing tool: A framework to evaluate quality of tests cases. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, ICSM '12, pages 646–649, Washington, DC, USA, 2012. Inst. de Tecnol. y Sist. de la Informacion, Univ. of Castilla-La Mancha, Ciudad Real, Spain, IEEE Computer Society.

[74] Andreas Zeller. *Why Programs Fail: A Guide to Systematic Debugging*. Morgan Kaufmann, Burlington, MA, 2 edition, 2009.

[75] Fanping Zeng, Liangliang Mao, Zhide Chen, and Qing Cao. Mutation-Based Testing of Integer Overflow Vulnerabilities. In *Wireless Communications, Networking and Mobile Computing, 2009. WiCom '09. 5th International Conference on*, WiCOM'09, pages 4416–4419, Piscataway, NJ, USA, 2009. Dept. of Comput., Univ. of Sci. & Technol. of China, Hefei, China, IEEE Press.

[76] Lingming Zhang, M Gligoric, D Marinov, and S Khurshid. Operator-based and random mutant selection: Better together, 2013.

[77] Chixiang Zhou and Phyllis Frankl. JDAMA: Java Database Application Mutation Analyser. *Softw. Test. Verif. Reliab.*, 21(3):241–263, September 2011.