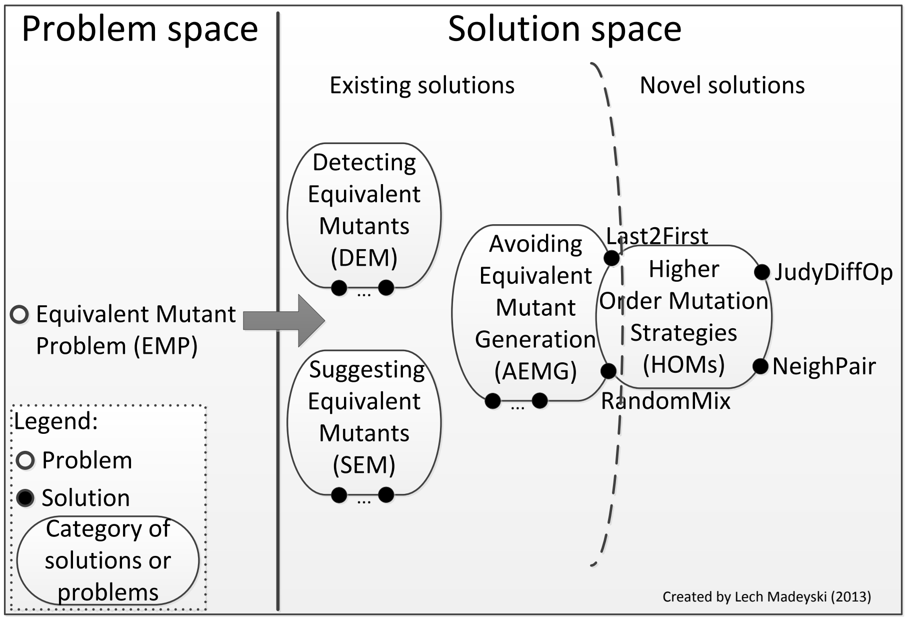
Overcoming the Equivalent Mutant Problem: A Systematic Literature Review and a Comparative Experiment of Second Order Mutation

Lech Madeyski, *Member, IEEE,* Wojciech Orzeszyna, Richard Torkar, *Member, IEEE,* and Mariusz Józala


Problem space
Solution space
Existing solutions
Novel solutions
Detecting Equivalent Mutants (DEM)
Suggesting Equivalent Mutants (SEM)
Avoiding Equivalent Mutant Generation (AEMG)
Higher Order Mutation Strategies (HOMs)
Last2First
JudyDiffOp
NeighPair
RandomMix
Equivalent Mutant Problem (EMP)
Legend:
Problem
Solution
Category of solutions or problems
Created by Lech Madeyski (2013)

Abstract—
Context. The equivalent mutant problem (EMP) is one of the crucial problems in mutation testing widely studied over decades.
Objectives. The objectives are: to present a systematic literature review (SLR) in the field of EMP; to identify, classify and improve the existing, or implement new, methods which try to overcome EMP and evaluate them.
Method. We performed SLR based on the search of digital libraries. We implemented four second order mutation (SOM) strategies, in addition to first order mutation (FOM), and compared them from different perspectives.
Results. Our SLR identified 17 relevant techniques (in 22 articles) and three categories of techniques: detecting (DEM); suggesting (SEM); and avoiding equivalent mutant generation (AEMG). The experiment indicated that SOM in general and *JudyDiffOp* strategy in particular provide the best results in the following areas: total number of mutants generated; the association between the type of mutation strategy and whether the generated mutants were equivalent or not; the number of not killed mutants; mutation testing time; time needed for manual classification.
Conclusions. The results in the DEM category are still far from perfect. Thus, the SEM and AEMG categories have been developed. The *JudyDiffOp* algorithm achieved good results in many areas.

Index Terms—mutation testing, equivalent mutant problem, higher order mutation, second order mutation.

1 Introduction

Mutation testing is a fault-based technique which measures the fault-finding effectiveness of test suites, on the basis of induced faults [12], [21]. Mutation testing induces artificial faults or changes into an application (mutant generation) and checks whether a test suite is "good enough" to detect them. However, there are mutations which keep the program semantics unchanged and thus cannot be detected by any test suite. The prob-

- *Lech Madeyski is with the Institute of Informatics, Wroclaw University of Technology, Poland. E-mail: lech.madeyski@pwr.wroc.pl. WWW: http://madeyski.e-informatyka.pl/*
- *Wojciech Orzeszyna is with the Institute of Informatics, Wroclaw University of Technology, Poland and the Blekinge Institute of Technology, Sweden.*
- *Richard Torkar is with the Chalmers and the University of Gothenburg, and Blekinge Institute of Technology, Sweden.*
- *Mariusz Józala is with the Institute of Informatics, Wroclaw University of Technology, Poland.*

Digital Object Identifier no. 10.1109/TSE.2013.44

lem of detecting equivalence either between two arbitrary programs or two mutants is an undecidable problem [4], [8], [18], [56], [64] and is known as the equivalent mutant problem (EMP).

Mutation testing provides a "mutation score" (MS), or "mutation adequacy", which is a testing criterion to measure the effectiveness or ability of a test suite to detect faults [9], [12], [21], [79]:

$$MS = \frac{M_K}{M_T - M_E} \tag{1}$$

Where M_T is the total number of produced mutants, M_K is the number of *killed mutants* (where the difference in behaviour between the original program and the mutated one was observed) and M_E is the number of equivalent mutants.

There is a range of mutation testing tools like Judy [52], Javalanche [71] or μJava (MuJava) [46] with MuClipse [73]. Unfortunately, finding equivalent mutants still consumes a lot of time and there is no automated way to detect all of the equivalent mutants. Furthermore, as observed by Schuler and Zeller [72], it takes an average of 15 minutes to assess one single mutation for equivalence. Therefore, analysing real world software projects there is often a need (also in this paper) to ignore equivalent mutants, which would mean that we can only measure the mutation score indicator [47], [48], [49], [52]:

$$MSI = \frac{M_K}{M_T} \tag{2}$$

It is still a valuable measure but not as desirable as obtaining the mutation score (Equation 1).

The rest of this paper is organized as follows: Section 2 presents a systematic literature review of equivalent mutant detection methods. The results of the systematic review (Section 3) indicate that the most promising techniques for handling EMP is higher order mutation (HOM) in general, and second order mutation (SOM) in particular. SOM testing strategies found in the systematic review are presented in detail in Section 4. Those strategies were analysed, extended, improved, and implemented in the Judy mutation testing tool [52] and then empirically evaluated on a number of open source software items. Section 5 presents the design and execution of the experiment, while Section 6 presents the results of the analysis concerning various SOM and FOM testing strategies. Threats to validity are discussed in Section 7, while Section 8 discusses the results. Conclusions and future work are presented in Section 9.

2 Systematic review

The authors followed the instructions presented by Kitchenham et al. [40]. As far as we know, there is no previous systematic literature review (SLR) regarding the equivalent mutant problem. The only study which can be classified as a systematic review is the paper by Jia and Harman [31], which focused on mutation testing in general and not on the EMP. In their insightful work, Jia and Harman only mentioned some of the most crucial methods and, thus, the relevant research questions posed by us could not be answered by their review.

Most of the papers found in our preliminary search, e.g. [60], [70], [75], include sections such as "Related work," where the authors discuss some of the existing approaches. However, they do not perform an SLR and, thus, only a small number of the existing methods are introduced in an *ad hoc* manner. We were, therefore, not convinced that a representative sample had been presented previously.

The protocol of our systematic literature review is publicly available online [62].

2.1 Research questions

Research questions must determine the goal of an SLR [7], [38], [40], [41], [43]. The objective of this study was to find a method (or methods) with which we would be able to overcome the equivalent mutant problem to a possibly most significant extent.

- **RQ1: Which of the existing methods try to solve the problem of equivalent mutants?**
  This is a very general question. In this case general ideas are also expected. Some of them might have been implemented and evaluated, while some might be theoretical suggestions for further refinements.
- **RQ2: How can those methods be classified?**
  As a result, the classification of the existing methods to some general domains and areas is expected.
- **RQ3: What is the maturity of the existing methods?**
  All existing methods will be grouped by their maturity.
- **RQ4: What are the theoretical ideas on how to improve the techniques which have already been empirically evaluated?**
  In this case, all the sources which the authors mention in "Future work" are to be analysed. Any possible suggestions which would lead to an increase in the number of detected equivalent mutants are welcome.

2.2 Search terms

For each research question, related major terms were developed. Synonyms, variations in spelling and structure (e.g. terms with and without hyphenation) were considered and accounted for in the queries formed. After constructing the preliminary search strings, pilot testing against the search engines was also undertaken in order to investigate the capability of the search engines, e.g. the handling of Boolean combinations and sub-query nesting. The resulting query was as follows:

equivalen* AND mutant* AND (mutation OR testing OR analysis OR problem* OR issue* OR question* OR (detect* OR find* OR recognize* OR catch*) AND (method* OR technique*) OR (method* OR technique*) AND (classification* OR ranking* OR classified OR categorisation* OR categorization* OR systematisation OR type* OR kind*) OR (method* OR technique*) AND (empirical*

OR evaluat* OR implement* OR development OR developed) OR (method* OR technique*) AND (further OR next OR future OR new) OR (method* OR technique*) AND (improv* OR progress* OR enhanc* OR refin* OR increas*))

The detailed forms (due to differences in search capabilities between various databases) are presented in the SLR protocol [62]. The title, abstract and keywords of the articles in the electronic databases were searched according to those search terms.

2.3 Resources to be searched

2.3.1 Database search

The main information sources to be searched, in the first iteration, were electronic databases: the ACM Digital Library, IEEE Xplore, Science Direct, the Springer Link and the Wiley Online Library.

Those databases were selected because they had been used as sources for other reviews in this area [31]. Also, we had a number of "key papers" [6], [20], [25], [56], [58], [59], [72] and we verified that we could find all of them in the above databases.

2.3.2 Grey literature

In order to cover grey literature (not necessarily peer-reviewed) [68], some alternative sources were investigated:

- Google scholar
  We used three search terms for the first phase, and for each of them checked the first 200 results. The search terms were slightly modified in order to adopt them to Google scholar and to improve the effectiveness of the search process. We used the following search terms:
  - equivalen* AND mutant* AND (mutation OR testing OR analysis)
  - equivalen* AND mutant* AND (method* OR technique*)
  - equivalen* AND mutant* AND (problem* OR issue* OR question*)
- All the proceedings from *"Mutation: The International Workshop on Mutation Analysis"* (five editions: 2000–2010).
- Scanning lists of references in all primary studies[1], according to the snowball sampling method [19].
- Checking the personal websites of all the authors of primary studies, in search of other relevant sources (e.g. unpublished or latest results).
- Contacting all the authors of primary studies. The authors were contacted in order to make sure that no relevant material had been missed.

It is also worth mentioning that the Mutation Testing Repository [32] provides a very thorough coverage of the publications in the literature on Mutation Testing and, therefore, is a highly recommended resource.

1. The research papers summarised in the review are referred to as primary studies, while the review itself is a secondary study [7]

2.4 Results selection process

The following inclusion criteria were taken into account when selecting the primary studies (it was enough for the paper to pass one of them):

- Describes at least one method for detecting, suggesting or avoiding equivalent mutants (this could include proof of concepts, empirically evaluated solutions, as well as theoretical ideas).
- Discusses the classification of the aforementioned methods.
- Evaluates, analyses or compares the aforementioned methods.
- Determines the current state of maturity of the methods dealing with EMP (theoretical ideas/proofs of concept/empirically evaluated solutions).
- Proposes theoretical ideas on how to improve the already evaluated methods dealing with EMP.

If the analysed study referred to one of the previously selected primary studies, then it additionally drew our attention but it was not the inclusion criterion per se.

The following type of studies were excluded (exclusion criteria):

- The article's language was other than English.
- The full text of the article could not be found.
- The article concerned mutations in the fields of study other than software engineering or computer science, e.g. genetics.

2.5 Quality assessment

In addition to the general inclusion and exclusion criteria, it is important to assess the quality of primary studies [38]. Study quality assessment was adopted in order to determine the strength of the evidence and to assign grades to the recommendations generated by the systematic review [34]. The questionnaire used in this study was based on the recommendations by Kitchenham and Charters [40] and Khan et al. [34] with some specific additions resulting from our research questions. The quality assessment questionnaire can be found in the SLR protocol [62].

3 Review results

A detailed process of identifying relevant literature is presented in Figure 1. The number of results found and used in each phase of the SLR are shown in Figure 1. In the end, we found 22 primary studies. One of them [59] was a substantial extension of the earlier conference paper [56]. All of the primary studies, except for one [31], presented methods for how to deal with the equivalent mutant problem. The exception, Jia and Harman's [31] study, is a valuable survey of the development of mutation testing, which, however, only lists and briefly describes some crucial approaches.

Below we have ranked the top-5 authors, according to the number of publications. The most active researchers in the subject of EMP were thus:

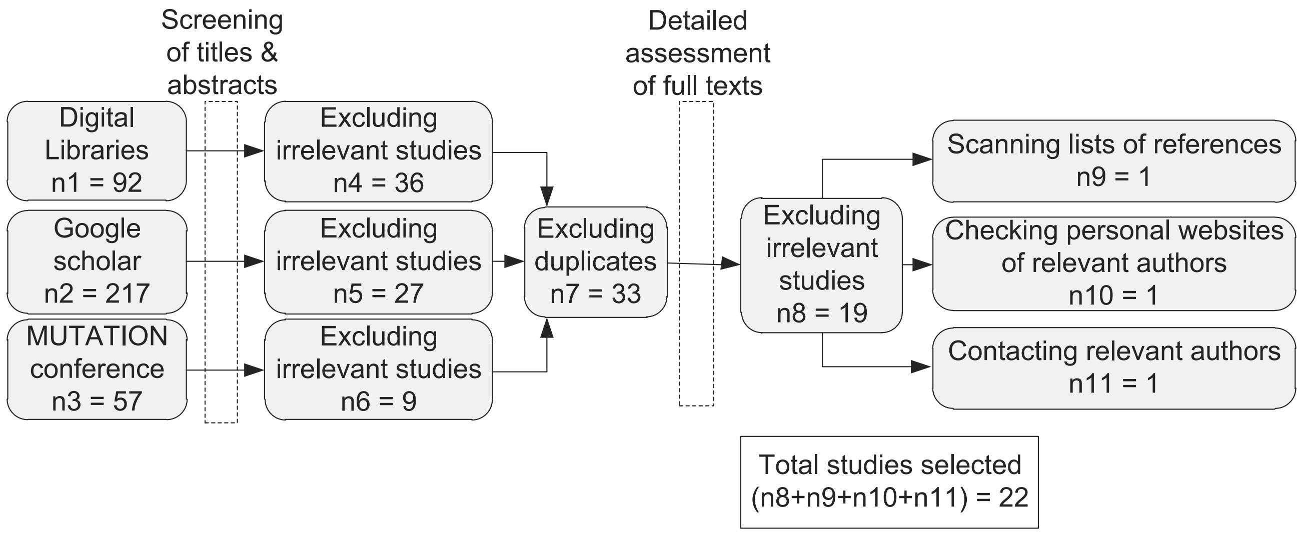

Screening of titles & abstracts
Detailed assessment of full texts
Digital Libraries n1 = 92
Google scholar n2 = 217
MUTATION conference n3 = 57
Excluding irrelevant studies n4 = 36
Excluding irrelevant studies n5 = 27
Excluding irrelevant studies n6 = 9
Excluding duplicates n7 = 33
Excluding irrelevant studies n8 = 19
Scanning lists of references n9 = 1
Checking personal websites of relevant authors n10 = 1
Contacting relevant authors n11 = 1
Total studies selected (n8+n9+n10+n11) = 22

Fig. 1. Identifying relevant literature.

- M. Harman (University College London, UK) [1], [22], [25], [30], [31]
- J. Offutt (George Mason University, USA) [56], [57], [58], [59], [61]
- R. Hierons (Brunel University, UK) [1], [22], [25]
- D. Schuler (Saarland University, Germany) [20], [70], [72]
- A. Zeller (Saarland University, Germany) [20], [70], [72]

The literature published by those authors represent 55% of all primary studies.

In the following section, each of the previously presented research questions is examined separately with the help of the findings from the SLR (see Table 1).

3.1 Which of the existing methods try to solve the problem of equivalent mutants?

On the basis of the primary studies, we have found 17 methods for equivalent mutant detection (in chronological order):

- Compiler optimizations techniques [6], [58] (1979)
- Using mathematical constraints to automatically detect equivalent mutants [56], [59] (1996)
- Using program slicing to assist in the detection of equivalent mutants [25] (1999)
- Selective mutation [55] (1999)
- Avoiding equivalent mutant generation using program dependence analysis [22] (2001)
- Using Bayesian-learning based guidelines to help to determine equivalent mutants [76] (2002)
- Co-evolutionary search techniques [1] (2004)
- Using equivalency conditions to eliminate equivalent mutants for object oriented mutation operators [61] (2006)
- Using semantic differences in terms of a running profile to detect non-equivalent mutants [17](2007)
- Margrave's change-impact analysis [54] (2007)
- Using Lesar model-checker for eliminating equivalent mutants [15] (2008)
- Examining the impact of equivalent mutants on coverage [20] (2009)
- Distinguishing the equivalent mutants by semantic exception hierarchy [28](2009)
- Higher order mutation testing [30], [37], [57], [63] (2009)
- Using a fault hierarchy to improve the efficiency of the DNF logic mutation testing [33] (2009)
- Using the impact of dynamic invariants [70] (2009)
- Examining changes in coverage to distinguish equivalent mutants [69], [72] (2010)

The paper's length limit does not allow us to explain the details of the aforementioned methods, as the number of the latter is quite large. The details of the methods, however, are described in the references we cite above (each of the techniques found in this review has at least one reference). The readers interested in the basics of software testing in general, and mutation testing in particular, are expected to look through books [2], [78] which complement the above mentioned references and extend the coverage of the topic.

Figure 2 shows how the primary studies are distributed according to programming language implementation. Java, Fortran and C are the three languages with the highest rank. Early work on dealing with equivalent mutants (including some avoidance rules) were carried out using Fortran [36]. For C programs, the tools Proteum [11], MILU [29] or Csaw [17] were used; while for Java programs, it was muJava [46] and the more recent Javalanche [71] and Judy [52]. There were no publications describing solutions for the EMP applied in C# and C++. For example in CREAM [13], [14] (a mutational tool for C#), equivalent mutants were identified by hand.

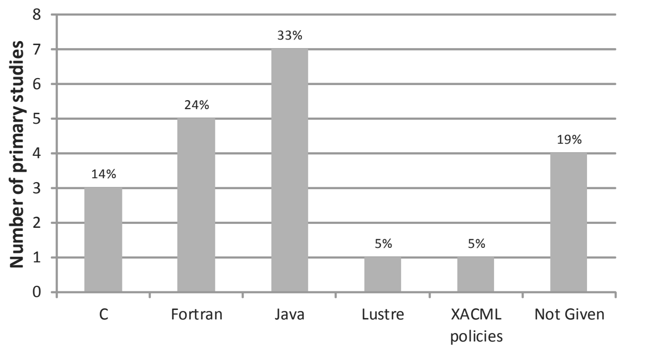


Fig. 2. Percentage of primary studies (methods only) addressing the equivalent mutant problem in different programming languages.

3.2 How can the equivalent mutant detection methods be classified?

There was no classification of equivalent mutant detection methods proposed in any of the papers found. Even the latest analysis and survey of the development of mutation testing by Jia and Harman [31] has not provided any categorization yet, only a list of some important approaches in chronological order. Due to the amount of research in this field, some sort of early classification would be helpful to summarize existing the techniques and to help indicate future work.

Two approaches to grouping have been considered. The first one - according to the application domain of the proposed solution (e.g. compiler optimization techniques or co-evolutionary approaches) and the second - according to the character of the obtained results (e.g. direct indication, suggestion, etc.) During the data extraction process we have noticed that almost all of the included techniques in the primary studies have their origins in different and unique areas of computer science. With such a classification, we would get many categories containing primarily one or, at the most, two methods. (That might be a good solution after some time, if this area continues to grow.) Moreover, as shown in the answer to the fourth question, the majority of the proposed approaches combine solutions from more than one field of study. Therefore, we would like to propose the classification of methods based on the collected data (especially as found in the column "Method Effectiveness" in Table 1).

Our classification distinguishes between three main categories of methods used to overcome the EMP (for sake of brevity when there is more than one publication discussing the topic we have sometimes introduced a new description of a method):

1) Detecting equivalent mutants techniques
   - Compiler optimizations techniques [6], [58] (1979)
   - Mathematical constraints to automatically detect equivalent mutants [56], [59] (1996)
   - Program slicing to assist in the detection of equivalent mutants [25] (1999)
   - Semantic differences in terms of running profile to detect non-equivalent mutants [17](2007)
   - Margrave's change-impact analysis [54](2007)
   - Lesar model-checker for eliminating equivalent mutants [15] (2008)
2) Avoiding equivalent mutant generation techniques
   - Selective mutation [55] (1999)
   - Avoiding equivalent mutant generation using program dependence analysis [22] (2001)
   - Co-evolutionary search techniques [1] (2004)
   - Equivalency conditions to eliminate equivalent mutants for object-oriented mutation operators [61] (2006)
   - Fault hierarchy to improve the efficiency of DNF logic mutation testing [33] (2009)
   - Distinguishing the equivalent mutants by semantic exception hierarchy [28](2009)
   - Higher order mutation testing [30], [37], [57], [63] (2009)
3) Suggesting equivalent mutants techniques
   - Using Bayesian-learning based guidelines to help to determine equivalent mutants [76] (2002)
   - Examining the impact of equivalent mutants on coverage [20] (2009)
   - Using the impact of dynamic invariants [70] (2009)
   - Examining changes in coverage to distinguish equivalent mutants [69], [72] (2010)

Figure 3 shows the distribution of primary studies over the years. It is quite clear that recently researchers have focused more on two categories of methods to overcome the EMP: avoiding equivalent mutant generation and suggesting equivalent mutants. We can only speculate as to the reason behind that tendency, but a plausible explanation is that detection techniques are also very hard to implement, and few researchers in the past decade have tackled testing problems which require hard programming. Beyond Offutt's research on software testing coupling effects and higher order mutation testing from 1992 [57], which actually was not focused on the EMP *per se* (it was not considered to be the main benefit of this technique), we can claim that the first and most obvious way of dealing with equivalent mutants are the equivalent mutant detection techniques (the first category). The most effective approach from this category detects 47.63% of the equivalent mutants and finds over 70% of unreachable statements [56], [59]; however, such a solution still needs a lot of manual and error-prone work. An advantage of detecting techniques is that they give no false positives, as suggesting equivalent mutants does. On the other hand, detecting techniques can never be complete. In summary, all three categories are thus complementary.

With the beginning of the 21st century, two new approaches began to be considered. Ever since then, the best method for suggesting equivalent mutants to a software tester has been considered to be the technique of assessing the impact of a mutant's internal behaviour as proposed

TABLE 1
Primary studies.

Ref	Authors	Year	Language	Maturity	Method Effectiveness	QA Result
[6]	D. Baldwin, F. Sayward	1979	Fortran	TI	Not Given	72%
[57]	J. Offutt	1992	Fortran	EE	Only from 0.53% to 1.4% generated 2-order mutants were equivalent.	82%
[58]	J. Offutt, M. Craft	1994	Fortran	EE	About 10%, with 25% standard deviation	100%
[56], [59]	J. Offutt, J. Pan	1996	Fortran	EE	47.63%	100%
[25]	R. Hierons et al.	1999	Not Given	TI	Should be equal to constraint solving approaches [56], [59] — 47.63%. However, slicing was able to subsume the Offut&Pan's approach [56], [59] and find additional equivalent mutants.	89%
[55]	E. Mresa, L. Bottaci	1999	Fortran	EE	Not Given	85%
[22]	M. Harman et al.	2001	Not Given	TI	Not Given	89%
[76]	A. Vincenzi et al.	2002	C	EE	Not Given	100%
[1]	K. Adamopoulos et al.	2004	Not Given	TI	Avoids equivalent mutant generation	72%
[61]	J. Offutt et al.	2006	Java	EE	Not Given	95%
[17]	M. Ellims et al.	2007	Not Given	TI	Not Given	75%
[54]	E. Martin, T. Xie	2007	XACML policies	PoC	Not Given	78%
[15]	L. du Bousquet, M. Delaunay	2008	Lustre	EE	Not Given	95%
[20]	B. Grün et al.	2009	Java	EE	Suggests (non-)equivalent mutants	100%
[28]	C. Ji et al.	2009	Java	TI	100%	78%
[30]	Y. Jia, M. Harman	2009	C	PoC	Not Given	83%
[33]	G. Kaminski, P. Ammann	2009	Java	EE	Avoids equivalent mutant generation	85%
[70]	D. Schuler et al.	2009	Java	EE	Not Given	100%
[37]	M. Kintis et al.	2010	Java	EE	Reduces the number of generated equivalent mutants (from 65.6% to 86.8%).	91%
[63]	M. Papadakis, N. Malevris	2010	C	EE	Reduction of approx. 80% to 90% of generated equivalent mutants	91%
[69], [72]	D. Schuler, A. Zeller	2010	Java	EE	Suggests non-equivalent mutants with a 75% probability	91%
[31]	Y. Jia, M. Harman	2010	-	-	-	67%

TI - Theoretical idea; PoC - Proof of concept; EE - Empirically evaluated

by Schuler [72]. If it is observed that the mutation changes coverage, it has a 75% chance of being non-equivalent.

From the group of techniques which avoid equivalent mutant generation, two recent studies provide interesting results. Both of the papers are empirical evaluations of higher order mutation testing. The method implemented by Papadakis and Malevris [63] for the C programming language leads to the reduction of approximately 80–90% of the generated equivalent mutants. For the Java language, according to Kintis et al. [37], the obtained results vary from 65.5% for $HDom(50\%)$ to 86.8% for the $SDomF$ strategy with the loss of test effectiveness being only 1.75% for $HDom(50\%)$ and 4.2% for $SDomF$[2].

As Table 1 indicates, only a small number of studies provide explicit results, which, thus, makes it difficult to compare methods.

2. $HDom(50\%)$ and $SDomF$ are the names of the mutation testing strategies evaluated by Kintis et al. [37]. It is worth mentioning that 50% in the name of the former strategy comes from the fact that besides SOMs generated by the strategy on a basis of FOMs, a randomly selected subset of the 50% of the remaining FOMs is included in the generated set of mutants. Hence the HDom(50%) strategy produces both FOMs and SOMs.

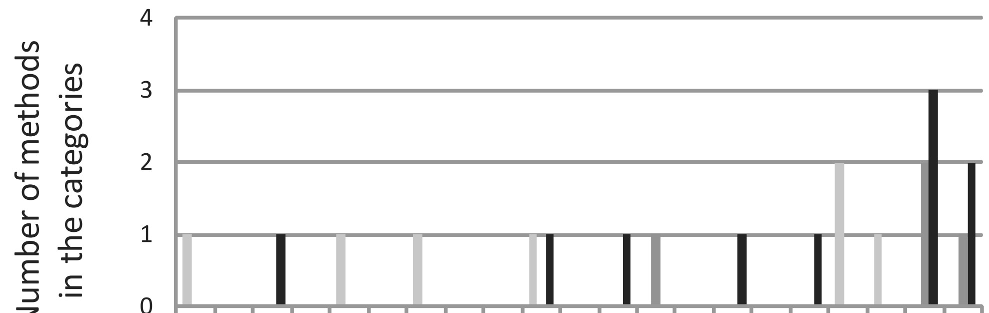
3.3 What is the maturity of existing methods?

In order to categorize further the identified methods we have distinguished between three categories: theory (six studies), proof of concept (two studies) and empirically evaluated methods (thirteen studies). In short, 62% of the studies are classified as being empirically evaluated.

Figure 4 shows the number of techniques by year (1979–2010). It is clear that the number of published studies in recent years is growing and most of the recent techniques are empirically evaluated. That provides some evidence corresponding with the results obtained for mutation testing in general by Jia and Harman [31] that EMP, like the overall field of mutation testing, is moving from theory to practical solutions.

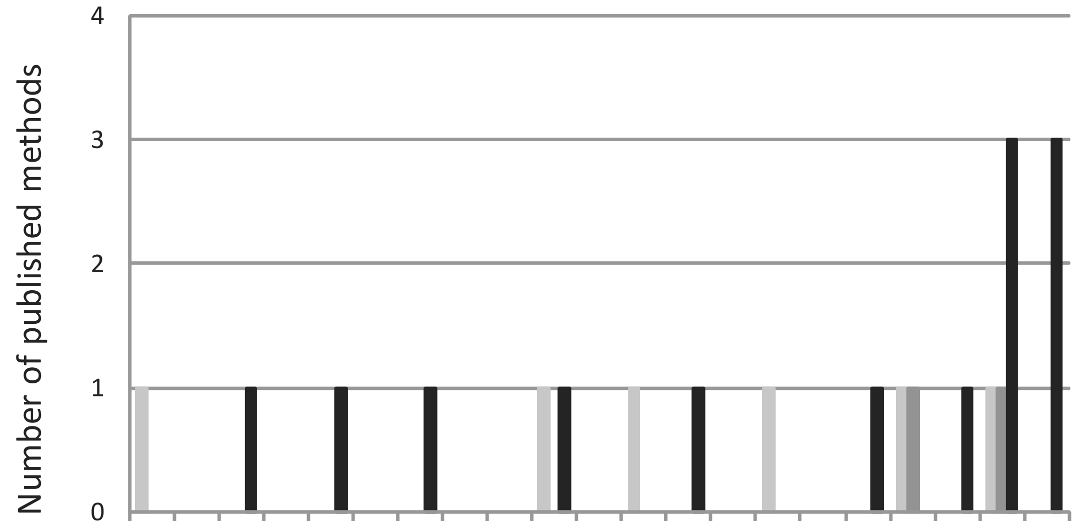
3.4 What are the theoretical ideas on how to improve already empirically evaluated techniques?

Seven out of thirteen (54%) publications which contain empirical evaluation present ideas on how to improve the proposed methods. Furthermore, the authors of three theoretical studies have also provided some ideas for future work, hence, a total of 50% of the *primary studies* suggests future improvements.



	79	…	92	93	94	95	96	97	98	99	00	01	02	03	04	05	06	07	08	09	10
▨ Detecting equivalent mutants	1	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	2	1	0	0
▨ Suggesting equivalent mutants	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	2	1
■ Avoiding equivalent mutant generation	0	0	1	0	0	0	0	0	0	1	0	1	0	0	1	0	1	0	0	3	2

Fig. 3. Classified solutions of equivalent mutant problem from 1992–2010 (cumulative view).



	79	…	92	93	94	95	96	97	98	99	00	01	02	03	04	05	06	07	08	09	10
▨ Theoretical idea	1	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	1	0	1	0
▨ Proof of concept	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
■ Empirically evaluated	0	0	1	0	1	0	1	0	0	1	0	0	1	0	0	0	1	0	1	3	3

Fig. 4. Solutions of the equivalent mutant problem by maturity from 1979–2010 (cumulative view).

Some authors have ideas on how to use two methods in tandem [22], [25], like for example Hierons et al. [25], in which the authors want to use a constraint solving technique together with program slicing. Other ideas are based on solving the problems which occurred in their specific studies [15], [56]. A very common suggestion is to consider some other possibilities [20], [28], [58], [70], [76], e.g. Schuler et al. [70] and Grün et al. [20] mention alternative impact measures, while Ji et al. [28] propose also to consider weak and firm mutations in higher order mutation testing.

3.5 Limitations of the review

This section presents the limitations of our SLR, in order to assess the validity of the outcome. The findings of this systematic review may have been mainly affected by the following limitations: difficulty in finding all the relevant studies (including grey literature); bias in the selection of the reviewed papers; inaccuracy in data extraction; inaccuracy in classifying the reported approaches; inaccuracy in assigning scores to each study of each element for the quality assessment criteria; and possible misinterpretations due to the fact that English is not the native language of the authors.

Finding all the relevant papers is known to be one of the major problems of systematic literature reviews [42]. In this case, we used an automated search of five main sources. However, we did not look into every possible source. The chosen databases were selected on the basis of the experiences shared by other groups [31], [41], [42].

Our search strings were designed to find the maximum number of known approaches towards EMP but it is still possible that we have left out the studies which might describe their subject in terms other than "equivalent mutant".

Due to the growing interest and the number of publications in the research area of mutation testing, some relevant papers may have been omitted. However, like other researchers of SLR, we are confident that it is not a systematic error [26], [35]

In addition, we found it to be a good practice to include grey literature search in order to make sure we covered non-published work to some extent [39]. For that reason, when considering grey literature, we took a large number of sources into account (e.g. Google Scholar, the personal webpages of authors and snowball sampling). The final step was to contact all relevant authors to review our list of primary studies.

In order to avoid subjective bias, it is helpful to follow the best practices suggested by the SLR practitioners. For example, it is recommended that three researchers be involved in the literature review process [5]. For both the screening and data extraction phases three people were involved in order to avoid subjective bias and to resolve doubts and discrepancies.

To ensure that the selection process during the detailed assessment of the papers' full text was entirely independent, we recorded the motivations for its inclusion or exclusion; then we verified the findings according to the inclusion and exclusion criteria from the SLR protocol. During full text screening we had some discrepancies, e.g. an article by Mresa and Botacci [55] was initially excluded by one of the researchers but we finally decided to include it due to the description of selective mutation from the perspective of the equivalent mutant problem.

The process of classifying the approaches towards EMP, as well as classifying the maturity (theoretical ideas, proofs of concept and empirically evaluated solutions), involved subjective decisions on the part of the researchers. To minimize these limitations, whenever there was a doubt on how to classify a particular paper, we discussed the case in order to resolve all discrepancies and doubts. During the data extraction phase we found several papers which lacked sufficient details regarding method effectiveness, i.e. in our sample of 22 papers only 8 papers provided details regarding the method's effectiveness. Due to that limitation, we were unable to compare methods and offer a complete view of their effectiveness.

Since English is not the native language of any of the researchers involved in this study, there is a risk that some of the papers have been misinterpreted during any of the stages of the performed literature review. On the other hand, all the decisions and results were checked by all of the authors.

3.6 Conclusions of the systematic review

The first part of the paper provided a detailed review of the EMP area. As has been shown, the last twenty years have witnessed a particularly large increase in the number of approaches on how to solve the EMP, with many of them in an advanced maturity stage.

So far, the paper has identified the existing methods for EMP and provided data in order to highlight the growth of the number of papers. The collected data also offer suggestions on how to improve these techniques. In addition, we have proposed a detailed categorization of the existing approaches, i.e. detecting, suggesting and avoiding equivalent mutant generation.

One contribution of our SLR, in comparison to Jia and Harman's survey [31], is a more complete list of the existing solutions for the equivalent mutant problem. With a thorough analysis of the available sources, including coming into contact with all relevant authors and scanning their personal websites, more methods have been identified. We have investigated avoiding equivalent mutant generation techniques as an additional group of approaches and found some omitted methods in other categories. Obviously, by focusing only on the equivalent mutant problem and having more delimited research questions, our study consequently supplies more detailed results from the EMP perspective. It is important to mention, though, that taking a subset of Jia and Harman's results regarding EMP will not give as complete a view on EMP as our SLR actually does.

The most promising technique for overcoming EMP seems to be higher order mutation (HOM) in general, and second order mutation (SOM) in particular. SOM has potential advantages to be of benefit for mutation testing tools, e.g. reducing the number of equivalent mutants [30], [57], [63] and reducing test effort (testing time) due to a reduced number of produced second order mutants [30], [63]. Furthermore, the manual assessment of mutant equivalence in the case of second order mutants should be fast. If the first of two first order mutants (combined to produce a second order mutant) is non-equivalent then it is very likely that the second order mutant will be non-equivalent too [65, Table I]. Hence, for the remaining part of the paper, we will focus on the SOM testing strategies, present implementations of the SOM strategies in the Judy mutation testing tool for Java [52], and empirically evaluate those implementations.

4 Higher order mutation testing strategies

Higher order mutation testing was initially introduced in the context of the mutant coupling effect in 1992 by Offutt. Offutt showed that "the set of test data developed for FOMs actually killed a higher percentage of mutants when applied to SOMs" [57].

Jia and Harman [30] distinguished between six types of HOMs and created a categorization of HOMs. They introduced the concept of subsuming and strongly subsuming higher order mutants (subsuming HOMs are harder to kill than FOMs from which they are constructed). The authors suggested that it might be preferable to replace constituent FOMs with a single HOM as a cost reduction technique. In particular, strongly subsuming HOMs are highly valuable to the mutation testing process. They are only killed by a subset of the intersection of test cases which kill each constituent FOM. As we may see via analysis of the types of HOMs discussed by Jia and Harman [30], there is no simple relation between killabilities (defined as how hard it is to kill the mutant) of HOM and the FOMs the HOM is constructed from. Jia and Harman also concluded that "the numbers of strongly subsuming HOMs is high and introduced a search-based optimization approach to select valuable HOMs" [30].

It is also worth mentioning that there is some empirical evidence to suggest that the majority of real faults are complex faults [16], [66]. A complex fault is a fault that cannot be fixed by making a single change to a source statement [57]. Such complex faults could only be simulated by higher order mutation [23]. The empirical results by Purushothaman and Perry [66] also reveal that there is less than 4 percent probability that a one-line change will introduce a fault in the code. All of those arguments make HOM an interesting alternative to evaluate to FOM.

Langdon et al. [44] applied multi objective Pareto optimal genetic programming to a generation of HOMs. Their algorithm evolves mutant programs according to two fitness functions: semantic difference and syntactic difference. In their experiment, they found realistic HOMs which are harder to kill as compared with first order mutants.

The opposite approach to selecting an optimal set of HOMs, according to the results from the mutation analysis of FOMs, is a technique used by Polo et al. [65]. They introduced three different algorithms (*Last2First*, *DifferentOperators*, and *RandomMix*) to combine FOMs to generate second order mutants (SOMs). Empirical results suggest that applying SOMs reduced the number of mutants by approximately 50%, without much decrease in the quality of the test suite.

Algorithms from the study by Polo et al. [65] were further investigated by Papadakis and Malevris [63], in particular, from the perspective of EMP. The results of their empirical study are promising: equivalent mutant reduction between 85.65–87.77% and fault detection ability loss from 11.45–14.57%. They indicate that SOMs can significantly decrease the number of introduced equivalent mutants and, because of approximately 50% mutants reduction, be a valid cost effective alternative.

Kintis et al. [37] presented another empirical study of higher order mutation testing strategies. They focused on the fact that SOMs achieve higher collateral coverage for strong mutation as compared with third or higher order mutants. A set of new SOM testing strategies was introduced and evaluated. The authors obtained the most promising results using hybrid strategies. Equivalent mutant reduction varied between 65.5% for $HDom(50\%)$ and 86.8% for the $SDomF$ strategy, with a loss of test effectiveness from just 1.75% for $HDom(50\%)$ to 4.2% for $SDomF$.

The short verbal description of algorithms given by Polo et al. [65] appears to be open to interpretation. As a result, there is no guarantee that our versions of the *Last2First* and *RandomMix* algorithms act in exactly the same manner as proposed by Polo et al. That sounds like a disadvantage but, fortunately, appears to be an advantage as well, because our version of the *DifferentOperators* algorithm (called *JudyDiffOp*) not only significantly differs from the original one but also outperforms Polo's et al. version. To help other researchers and practitioners replicate our study, we decided to include in the paper a detailed pseudo-code of the algorithms evaluated in our study (Algorithms 1-4).

The first algorithm proposed by Polo et al. [65] is the *Last2First* algorithm. It needs the list of first-order mutants in the order in which they were generated. *Last2First* combines the first mutant with the last, then the second with the next-to-last, and so on. Each first-order mutant is used once, except when the number of first-order mutants is odd. In that case, one mutant is used twice. The number of generated second-order mutants is reduced to half of the number of first-order mutants. The pseudo-code of the *Last2First* algorithm is presented as Algorithm 1.

Algorithm 1 *Last2First*(program, operators[ ]).

```
1: LET $firstOrderMutants$ be an empty list
2: FOR ALL $operator$ in $operators$
3:   $mutationPoints[] \Leftarrow operator.countMutationPoints(program)$
4:   FOR ALL $point$ in $mutationPoints$
5:     $possibleMutations[] = operator.countPossibleMutations(program, point)$
6:     FOR EACH $possibleMutant$ in $possibleMutations$ DO
7:       $newMutant \Leftarrow operator.mutate(program, point, possibleMutant)$
8:       $firstOrderMutants \Leftarrow newMutant$
9:     END FOR
10:  END FOR
11: END FOR
12: LET $secondOrderMutants$ be an empty list
13: WHILE $firstOrderMutants.size > 1$ DO
14:   $fom1 \Leftarrow firstOrderMutants \rightarrow last$
15:   $firstOrderMutants.remove(fom1)$
16:   $fom2 \Leftarrow firstOrderMutants \rightarrow first$
17:   IF $firstOrderMutants.size \neq 2$ THEN
18:     $firstOrderMutants.remove(fom2)$
19:   END IF
20:   $operator \Leftarrow fom2 \rightarrow operator$
21:   $newMutant \Leftarrow operator.mutate(fom1 \rightarrow program, fom2 \rightarrow point, fom2 \rightarrow possibleMutant)$
22:   $secondOrderMutants \Leftarrow newMutant$
23: END WHILE
24: RETURN $secondOrderMutants$
```

In the *DifferentOperators* strategy the combination of first-order mutants is made by selecting pairs that use mutants produced by different operators. The short verbal description of algorithms given by Polo et al. [65] leads to a situation where that can be interpreted differently. In the version implemented in the Judy mutation testing tool (hence called *JudyDiffOp*) each first-order mutant is used as little as possible (i.e. both constituent FOMs are used only once for producing a SOM). Our version gives at least a 50% mutants reduction and we find it hard to obtain the level of reduction as achieved by [63], [65]; however, it appears that our version outperforms Polo's version. The pseudo-code of the *JudyDiffOp* algorithm is presented as Algorithm 2.

Algorithm 2 *JudyDiffOp*(program, operators[ ]).

```
1: LET $firstOrderMutants$ be an empty list
2: FOR ALL $operator$ in $operators$
3:   $mutationPoints[] \Leftarrow operator.countMutationPoints(program)$
4:   FOR ALL $point$ in $mutationPoints$
5:     $possibleMutations[] = operator.countPossibleMutations(program, point)$
6:     FOR EACH $possibleMutant$ in $possibleMutations$ DO
7:       $newMutant \Leftarrow operator.mutate(program, point, possibleMutant)$
8:       $firstOrderMutants \Leftarrow newMutant$
9:     END FOR
10:  END FOR
11: END FOR
12: LET $secondOrderMutants$ be an empty list
13: WHILE $firstOrderMutants.size > 1$ DO
14:   $fom1 \Leftarrow firstOrderMutants \rightarrow first$
15:   $firstOrderMutants.remove(fom1)$
16:   WHILE $firstOrderMutants.size > 0$ DO
17:     $fom2 \Leftarrow firstOrderMutants \rightarrow first$
18:     IF $fom1 \rightarrow operator \neq fom2 \rightarrow operator$ THEN
19:       $operator \Leftarrow fom2 \rightarrow operator$
20:       $newMutant \Leftarrow operator.mutate(fom1 \rightarrow program, fom2 \rightarrow point, fom2 \rightarrow possibleMutant)$
21:       $secondOrderMutants \Leftarrow newMutant$
22:       $firstOrderMutants.remove(fom2)$
23:       BREAK
24:     END IF
25:   END WHILE
26: END WHILE
27: RETURN $secondOrderMutants$
```

RandomMix is the last algorithm from the set proposed by Polo et al. [65]. To allow for a comparison of the two previous algorithms with pure chance, that algorithm combines any two first-order mutants, using each mutant once. Similarly to *Last2First*, when the number of first-order mutants is odd, one of the mutants is used twice. By definition *RandomMix* reduces the number of generated second-order mutants by half, with respect to first-order mutants. The pseudo-code of the *RandomMix* algorithm is presented as Algorithm 3.

In contrast to the *Last2First* algorithm, we would like to introduce the *NeighPair* strategy. It combines FOMs which are as close to each other as possible, i.e. a list of mutation points for FOMs is created and neighbouring pairs are selected to construct SOMs. The number of generated SOMs is, thus, reduced by half. The pseudo-code of the *NeighPair* algorithm is presented as Algorithm 4.

It is also worth remembering that our SOM strategies

Algorithm 3 *RandomMix*(program, operators[ ]).

```
1: LET $firstOrderMutants$ be an empty list
2: FOR ALL $operator$ in $operators$
3:   $mutationPoints[] \Leftarrow operator.countMutationPoints(program)$
4:   FOR ALL $point$ in $mutationPoints$
5:     $possibleMutations[] = operator.countPossibleMutations(program, point)$
6:     FOR EACH $possibleMutant$ in $possibleMutations$ DO
7:       $newMutant \Leftarrow operator.mutate(program, point, possibleMutant)$
8:       $firstOrderMutants \Leftarrow newMutant$
9:     END FOR
10:  END FOR
11: END FOR
12: LET $secondOrderMutants$ be an empty list
13: WHILE $firstOrderMutants.size > 1$ DO
14:   $fom1 \Leftarrow firstOrderMutants \rightarrow first$
15:   $firstOrderMutants.remove(fom1)$
16:   $fom2 \Leftarrow firstOrderMutants \rightarrow random$
17:   IF $firstOrderMutants.size \neq 2$ THEN
18:     $firstOrderMutants.remove(fom2)$
19:   END IF
20:   $operator \Leftarrow fom2 \rightarrow operator$
21:   $newMutant \Leftarrow operator.mutate(fom1 \rightarrow program, fom2 \rightarrow point, fom2 \rightarrow possibleMutant)$
22:   $secondOrderMutants \Leftarrow newMutant$
23: END WHILE
24: RETURN $secondOrderMutants$
```

Algorithm 4 *NeighPair*(program, operators[ ]).

```
1: LET $firstOrderMutants$ be an empty list
2: FOR ALL $operator$ in $operators$
3:   $mutationPoints[] \Leftarrow operator.countMutationPoints(program)$
4:   FOR ALL $point$ in $mutationPoints$
5:     $possibleMutations[] = operator.countPossibleMutations(program, point)$
6:     FOR EACH $possibleMutant$ in $possibleMutations$ DO
7:       $newMutant \Leftarrow operator.mutate(program, point, possibleMutant)$
8:       $firstOrderMutants \Leftarrow newMutant$
9:     END FOR
10:  END FOR
11: END FOR
12: LET $secondOrderMutants$ be an empty list
13: WHILE $firstOrderMutants.size > 1$ DO
14:   $fom1 \Leftarrow firstOrderMutants \rightarrow first$
15:   $firstOrderMutants.remove(fom1)$
16:   $fom2 \Leftarrow firstOrderMutants \rightarrow first$
17:   IF $firstOrderMutants.size \neq 2$ THEN
18:     $firstOrderMutants.remove(fom2)$
19:   END IF
20:   $operator \Leftarrow fom2 \rightarrow operator$
21:   $newMutant \Leftarrow operator.mutate(fom1 \rightarrow program, fom2 \rightarrow point, fom2 \rightarrow possibleMutant)$
22:   $secondOrderMutants \Leftarrow newMutant$
23: END WHILE
24: RETURN $secondOrderMutants$
```

do not search for subsuming HOMs.

5 Experimental setup

The aim of the experiment was to answer the following research questions:

- **RQe1: What is the reduction in the number of mutants for the SOM strategies as compared with FOM?**
- **RQe2: What is the reduction in the number of equivalent mutants for the SOM strategies as compared with FOM?**
- **RQe3: What is the reduction in the number of live mutants for the SOM strategies as compared with FOM?**
- **RQe4: What is the relative change in mutation scores for each of the investigated SOM strategies as compared with FOM?**
- **RQe5: What is the reduction of mutation testing time using the SOM strategies as compared with FOM?**
- **RQe6: What is the potential reduction in the time required to assess whether each of the mutants is equivalent or non-equivalent?**

5.1 Software under test (SUT)

In most of the papers [23], [29], [30], [37], [44], [63] related to higher order mutant generation strategies the benchmark programs (SUT) were small (50–5,000 lines of code, or LOC). Only Polo et al. [65] applied their strategies to a SUT which had more than 10,000 lines of code. However, the most important concern regards the scalability of using mutation when we have thousands of classes. Solving that is as much about automation (by means of mutation testing tools which are able to smoothly integrate with different software development infrastructures) as about reducing the number of mutants (which we will discuss in the paper). Judy mutation testing tool for Java helped us to deal with both concerns.

For our experiment, we selected four open source projects, which are larger (in terms of lines of code) than those analysed by other researchers [23], [29], [30], [37], [44], [63], have high quality test cases and high branch coverage. We assumed that such programs would represent software developed in the industry and allow us to draw unbiased conclusions to some extent. Table 2 presents our software under test. Apart from the LOC, branch coverage, number of classes and test cases, we also included the mutation score indicator (MSI) [47], [48], [49], which is a quantitative measure of the quality of test cases, defined as the ratio of killed mutants to all mutants (see Equation 2).

This definition is different from mutation score (MS), as MSI ignores equivalent mutants. Hence, MSI can be seen as the lower bound on mutation score.

The following projects have been selected for the experiment:

- *Barbecue*[3] – is a library that provides the means to create barcodes for Java applications.
- *Apache Commons IO*[4] – is a library of utilities to assist with developing input/output functionality.
- *Apache Commons Lang*[5] – provides a host of helper utilities for the standard `java.lang` package, including operations on strings, collections, dates, etc.
- *Apache Commons Math*[6] – is a wide set of utilities for mathematical and statistical operations.

5.2 Supporting tool

For the experiment we have used Judy [52], a mutation testing tool for Java, which supports all three mutation testing phases: mutant generation, mutant execution and mutation analysis. We have extended the latest version of Judy [53] with second-order mutation testing mechanisms. The list and description of all 48 mutation operators available in Judy is presented in Table 3.

5.3 Experimental procedure

In the first phase we implemented all of the investigated strategies in Judy. Next, four 7–80 KLOC, open source programs were chosen (see Section 5.1) for an empirical evaluation. We first applied FOM testing on each SUT. In this way the number of all generated mutants, the number of all live mutants, and the MSI metric were obtained. Then, the comparison was performed with each SOM strategy. Each of the examined strategies were applied to each SUT, i.e. for four programs we applied five strategies (i.e. four SOM strategies as well as the FOM strategy).

To answer the second research question, all of the results were verified manually in order to identify possible equivalent mutants. However, determining the exact number of equivalent mutants was not the purpose of this study. In fact, it is a tedious and very time-consuming task [72], due to the large number of mutants in real world projects and the time necessary to assess whether each of the mutants is equivalent or non-equivalent (about 15 minutes according to Schuler and Zeller [72]). As the cost of manually collecting data for too many mutants is prohibitive, we needed to set a sample size out of convenience, i.e. we decided to manually analyse 50 randomly selected live mutants per strategy per SUT. As a result, we manually classified 1,000 mutants (five mutation strategies $\times$ four SUT $\times$ 50 mutants per sample) as equivalent or non-equivalent. During this process we kept in mind the characteristics of second order mutants' constituents, as introduced by Polo et al. [65, Table I].

The next section presents the comparisons regarding the reduction in the number of mutants (to answer RQe1), the reduction in the number of equivalent mutants (RQe2), the reduction in the number of live mutants (RQe3), the relative change in mutation scores (RQe4), the reduction

3. http://barbecue.sourceforge.net/
4. http://commons.apache.org/io/
5. http://commons.apache.org/lang/
6. http://commons.apache.org/math/

TABLE 2
Software under test.

Project	LOC	No. of classes	No. of test cases	Branch coverage	MSI	No. of FOMs
Barbecue	7,413	59	21	50%	41%	1112
Commons IO	16,283	100	43	84%	84%	5983
Commons Lang	48,507	81	88	89%	72%	17833
Commons Math	80,023	406	221	95%	77%	21691

TABLE 3
Mutation operators available in Judy mutation testing tool.

AIR	AIR_ Add	Replaces basic binary arithmetic instructions with ADD
	AIR_ Div	Replaces basic binary arithmetic instructions with DIV
	AIR_ LeftOperand	Replaces basic binary arithmetic instructions with their left operands
	AIR_ Mul	Replaces basic binary arithmetic instructions with MUL
	AIR_ Rem	Replaces basic binary arithmetic instructions with REM
	AIR_ RightOperand	Replaces basic binary arithmetic instructions with their right operands
	AIR_ Sub	Replaces basic binary arithmetic instructions with SUB
JIR	JIR_ Ifeq	Replaces jump instructions with IFEQ (IF_ ICMPEQ, IF_ ACMPEQ)
	JIR_ Ifge	Replaces jump instructions with IFGE (IF_ ICMPGE)
	JIR_ Ifgt	Replaces jump instructions with IFGT (IF_ ICMPGT)
	JIR_ Ifle	Replaces jump instructions with IFLE (IF_ ICMPLE)
	JIR_ Iflt	Replaces jump instructions with IFLT (IF_ ICMPLT)
	JIR_ Ifne	Replaces jump instructions with IFNE (IF_ ICMPNE, IF_ ACMPNE)
	JIR_ Ifnull	Replaces jump instruction IFNULL with IFNONNULL and vice-versa
LIR	LIR_ And	Replaces binary logical instructions with AND
	LIR_ LeftOperand	Replaces binary logical instructions with their left operands
	LIR_ Or	Replaces binary logical instructions with OR
	LIR_ RightOperand	Replaces binary logical instructions with their right operands
	LIR_ Xor	Replaces binary logical instructions with XOR
SIR	SIR_ LeftOperand	Replaces shift instructions with their left operands
	SIR_ Shl	Replaces shift instructions with SHL
	SIR_ Shr	Replaces shift instructions with SHR
	SIR_ Ushr	Replaces shift instructions with USHR
Inheritance	IOD	Deletes overriding method
	IOP	Relocates calls to overridden method
	IOR	Renames overridden method
	IPC	Deletes super constructor call
	ISD	Deletes super keyword before fields and methods calls
	ISI	Inserts super keyword before fields and methods calls
Polymorphism	OAC	Changes order or number of arguments in method invocations
	OMD	Deletes overloading method declarations, one at a time
	OMR	Changes overloading method
	PLD	Changes local variable type to super class of original type
	PNC	Calls new with child class type
	PPD	Changes parameter type to super class of original type
	PRV	Changes operands of reference assignment
Java-Specific Features	EAM	Changes an accessor method name to other compatible accessor method names
	EMM	Changes a modifier method name to other compatible modifier method names
	EOA	Replaces reference assignment with content assignment (clone) and vice-versa
	EOC	Replaces reference comparison with content comparison (equals) and vice-versa
	JDC	Deletes the implemented default constructor
	JID	Deletes field initialization
	JTD	Deletes this keyword when field has the same name as parameter
	JTI	Inserts this keyword when field has the same name as parameter
Jumble-Based [27], [74]	Arithmetics	Mutates arithmetic instructions
	Jumps	Mutates conditional instructions
	Returns	Mutates return values
	Increments	Mutates increments

of time required for mutation testing (RQe5) and the potential reduction in the time required to assess whether each of the second order mutants is equivalent or non-equivalent in comparison with first order mutants (RQe6).

6 Experimental results and analysis

The experimental results derived from the application of the FOM testing and the four SOM testing strategies are presented and analysed in this section.

6.1 Mutant reduction

For each of the analysed projects and investigated strategies, the number of generated first order mutations was compared with the number of produced second order mutations. The results are presented in Table 4. Decreasing the number of mutants (called mutants reduction) makes the process of mutation testing more efficient, since execution time decreases.

RandomMix, *Last2First* and *NeighPair* strategies achieved a reduction (approx. 50%) consistent with theory

TABLE 4
Total number of mutants using First Order Mutation (FOM) and different Second Order Mutation (SOM) strategies.

Project name	FOM	SOM strategies:			
		RandomMix	*Last2First*	*JudyDiffOp*	*NeighPair*
Barbecue	1,112	562	562	314	562
Commons IO	5,983	3,009	3,009	2,319	3,009
Commons Lang	17,833	8,930	8,930	6,452	8,930
Commons Math	21,691	10,498	10,498	9,252	10,498
Average reduction with respect to FOM:		50.2%	50.2%	63.5%	50.2%

(see Section 4), as well as the results from the studies of Polo et al. [65] and Papadakis and Malevris [63] (except, of course, for the *NeighPair* algorithm, which has not been previously evaluated). The highest mutant reduction was obtained by the *JudyDiffOp* strategy. The explanation is simple, the *JudyDiffOp* strategy by definition does not create SOMs from two consecutive FOMs, if the latter involve the same operator. Hence, the number of SOMs created according to the *JudyDiffOp* strategy is less, and the reduction is higher (i.e. well over 50%)..

It is worth noting that Papadakis and Malevris [63] only achieved a 27.68% reduction on average. This discrepancy in the results most likely stems from the differences in the implementations of the algorithms. In our version of the *DifferentOperators* algorithm (i.e. *JudyDiffOp*), both constituent FOMs were used only once for producing a SOM. This algorithm removes at least half the number of generated mutants. Consequently, with this version of *DifferentOperators* it is impossible to obtain mutant reductions at a level similar to Papadakis and Malevris [63]. Unfortunately, the authors of the original algorithm did not make available its code or pseudo-code, only a plain-text description, which is not precise enough to replicate their version of the algorithm. We can still declare with certainty that our modified version of the *DifferentOperators* algorithm (called *JudyDiffOp*) provides the highest mutants reduction. (We have included the pseudo-code of our implementation as Algorithms 1, 2, 3, and 4.)

On the basis of the empirical results (presented in this section) and statistical analysis of mutants reduction (described in detail in Appendix A1 [51]) one may come to the conclusion labelled as Finding 1.

Finding 1: The second order mutation strategy called *JudyDiffOp* significantly reduced the total number of generated mutants in comparison with the first order mutation. The size of the effect was large and in favour of *JudyDiffOp*.

Other findings are discussed in subsequent sections.

6.2 Equivalent mutant reduction

This section presents the achieved reductions of the introduced equivalent mutants. Two of the authors manually classified samples of live mutants. Following the experimental procedure described in Section 5.3, 1000 mutants were manually classified in total, i.e. 50 mutants for each of the analysed SUT (Barbecue, Commons IO, Commons Lang, Commons Math) and each of the analysed mutation strategy (FOM, *RandomMix* SOM, *Last2First* SOM, *JudyDiffOp* SOM, *NeighPair* SOM). The obtained results are shown in detail in Table 5. In each sample of 50 manually classified unkilled mutants in the SUTs, we found between 11 (in Commons Math) and 33 (in Barbecue) equivalent mutants using FOM, but only 7-9 equivalent mutants applying the *RandomMix* strategy, 5-6 equivalent mutants applying the *Last2First* strategy, 4-6 equivalent mutants applying the *JudyDiffOp* strategy, and 11-25 equivalent mutants applying the *NeighPair* strategy. Our results are in line with the results obtained by Schuler and Zeller [69], [72]. They found, by manual assessment of 140 uncaught mutations in seven Java programs, that 45% of all uncaught mutations were equivalent. We also agree with their explanation, which applies to our work as well, that this high number, although it may come as a surprise, comes from the fact that several non-equivalent mutants are already caught by the test suite.

Equivalent mutant reduction with respect to the FOM strategy is presented in Figure 5. This figure brings out more interesting findings and leads to four valuable conclusions. First of all, we should admit that all three strategies proposed by Polo et al. [65] reduce the number of equivalent mutants. By applying them to larger and more complex projects than in earlier publications, we provide even better indications of their value.

TABLE 5
Number of equivalent mutants in a sample (50 unkilled mutants) using First Order Mutation (FOM) and different Second Order Mutation (SOM) strategies.

Project name	FOM	SOM strategies:			
		Random Mix	*Last2 First*	*Judy DiffOp*	*Neigh Pair*
Barbecue	33	9	5	6	25
Commons IO	13	7	6	4	14
Commons Lang	12	7	6	5	14
Commons Math	11	8	6	5	11

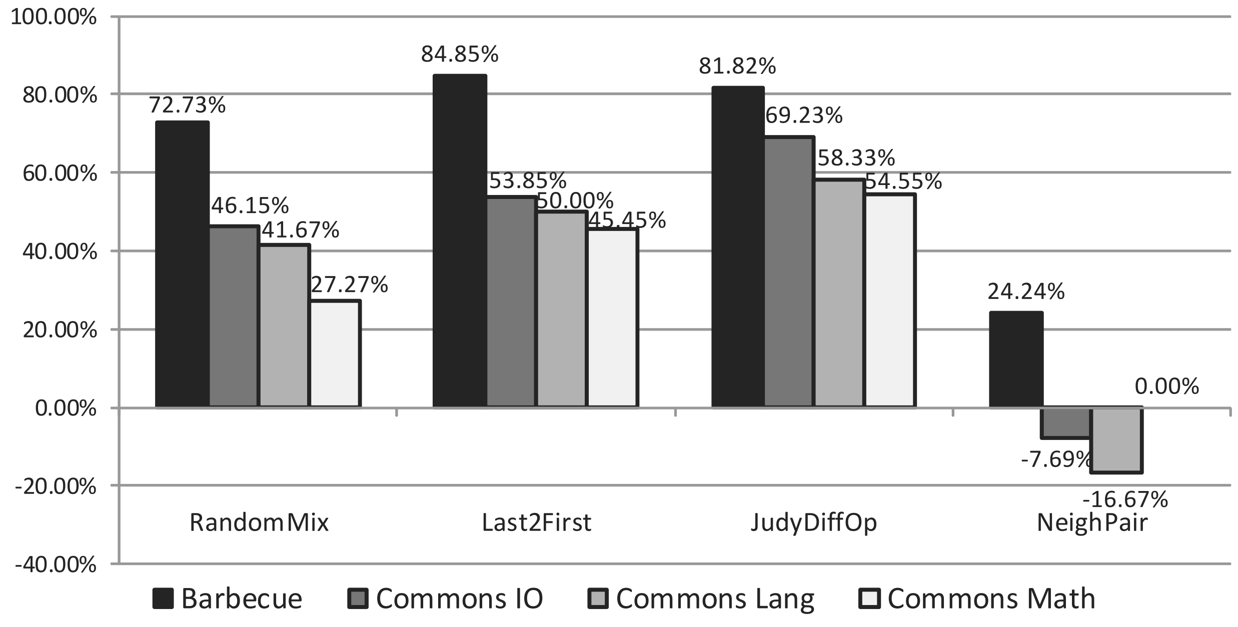


Fig. 5. Equivalent mutant reduction with respect to first order mutation.

For *NeighPair*—the new algorithm proposed by us—we obtained disappointing results. Equivalent mutant reduction was observable only for Barbecue. For the Commons IO and Lang projects, that strategy generated more equivalent mutants than the FOM strategy. From the perspective of equivalent mutant reduction, the *NeighPair* strategy generated the worst results.

Our second algorithm, *JudyDiffOp*, is based on the *DifferentOperators* idea by Polo et al. [65]. The algorithm generates the least equivalent mutants and from the perspective of the equivalent mutant problem seems to be the best choice. The results published by Papadakis and Malevris [63] are in contradiction with our results. In their study *Last2First* has the highest reduction (87.77%), followed by *RandomMix* (87.11%) and, finally, *DifferentOperators* (85.65%). In our study, the differences between reductions obtained for each strategy are not as small. A plausible explanation for the aforementioned differences comes from possible discrepancies between the textual descriptions provided by Polo et al. [65] and the algorithms which are described in the pseudo-code (Algorithms 1–4) and implemented by us.

The last conclusion is that the highest reduction was achieved for Barbecue; the smallest project (in terms of lines of code), with the lowest branch coverage and lowest MSI. The results obtained for Barbecue are close to the results presented by Papadakis and Malevris [63], who only analysed small projects (7 projects with a number of LOC below or equal to 513 and one project with LOC below 6 KLOC). They also achieved approximately 80% less equivalent mutants. It was observed that equivalent mutant reduction decreases with the increase in LOC (see Figure 6) or branch coverage (see Figure 7) for all the strategies except *NeighPair*.

The observed results are in line with our expectations (as the SOM approach *hides* equivalent mutants behind killable mutants or, looking from a different perspective, *excludes* equivalent mutants by combining them with the non-equivalent ones) but SOM strategies still differ between each other with regard to the number of equivalent mutants and this information can be of practical importance.

On the basis of the empirical results (presented in this section) and statistical analysis of the equivalent mutants reduction (described in detail in Appendix A2 [51]) one may come to the conclusion labelled as Finding 2.

Finding 2: The second order mutation significantly reduced the number of equivalent mutants in comparison to the first order mutation. The size of the effect was medium [67].

Other findings, e.g. related to the loss in testing strength, are discussed in subsequent sections.

6.3 Live mutant reduction

Table 6 presents the numbers of not killed (live) mutants which had to be classified as equivalent or non-equivalent. It is fairly easy to observe that the *JudyDiffOp* exhibits the best results among these four algorithms.

On the basis of the empirical results (presented in this section) and statistical analysis of the live mutants reduction (described in detail in Appendix A3 [51]) one may come to the conclusion labelled as Finding 3.

Finding 3: The second order mutation strategy, called *JudyDiffOp*, significantly reduced the number of

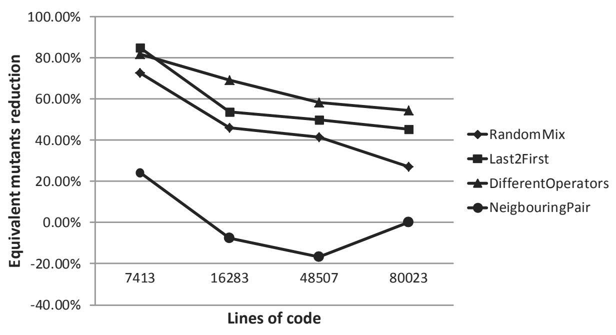


Fig. 6. The ratio of equivalent mutant reduction to lines of code in the project.

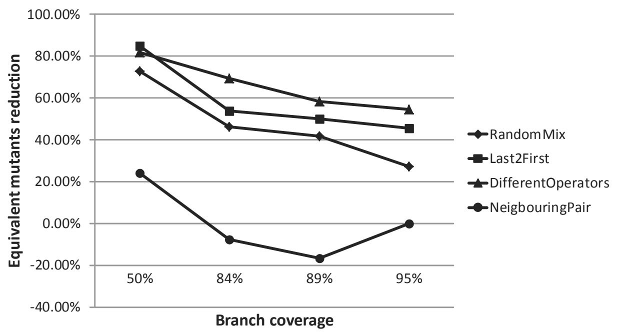


Fig. 7. The ratio of equivalent mutant reduction to the project's branch coverage.

TABLE 6
Number of live (i.e. not killed) mutants in the population using First Order Mutation (FOM) and different Second Order Mutation (SOM) strategies.

Project name	**FOM**	**SOM strategies:**			
		Random Mix	*Last2 First*	*Judy DiffOp*	*Neigh Pair*
Barbecue	661	289	282	80	303
Commons IO	972	296	274	262	446
Commons Lang	4973	979	945	699	1404
Commons Math	4974	664	932	753	1115

not killed mutants in comparison with the first order mutation. The size of the effect was large.

The magnitude of the observed effect is an indicator of its practical importance which, in turn, comes from the extremely high cost of manual classification of not killed mutants (as equivalent or non-equivalent).

6.4 Relative change in mutation score estimations

We measured the relative change in mutation score estimations ($RCMSE$) for each SOM (HOM) strategy in comparison to FOM. It allowed us to compare the FOM and the SOM results in mutation score estimations employing results from manual classification of 1000 live mutants as equivalent or non-equivalent. The detailed results of the manual classification of live mutants are presented in Section 6.2, while the experimental procedure (including sampling strategy) is described in Section 5.3.

The basic terms needed to define our $RCMSE$ metric (the relative change in mutation score estimations) are as follows: M_K is the number of *killed mutants* in the analysed mutation strategy (FOM or SOM), M_T is the total number of produced mutants (i.e. killed and live mutants added up: $M_K + M_L$) in the analysed mutation strategy (FOM or SOM) and $\widehat{M_E}$ is the estimated number of equivalent mutants in the analysed mutation strategy (FOM or SOM). The estimated number of equivalent mutants ($\widehat{M_E}$) comes from the number of live mutants (M_L) and the ratio of equivalent mutants in the manually classified sample ($R_{Esample}$). The ratio of equivalent mutants in the manually classified sample is defined as follows: $R_{Esample} = \frac{M_{Esample}}{SampleSize}$, where $M_{Esample}$ is the number of equivalent mutants in the manually classified sample, while $SampleSize$ is the number of mutants in the each manually classified sample, i.e. fifty mutants per each of the analysed projects and each of the analysed mutation strategy.

Let $\widehat{MS}_{FOM}$ and $\widehat{MS}_s$ be the estimations of the mutation score for the FOM and SOM strategies, respectively, obtained from the following equation

$$\begin{aligned}\widehat{MS} &= \frac{M_K}{M_T - \widehat{M_E}} \\ &= \frac{M_K}{M_T - (M_A \times R_{Esample})} \\ &= \frac{M_K}{M_T - (M_L \times \frac{M_{Esample}}{SampleSize})}\end{aligned} \tag{3}$$

Having all the basic ingredients defined we may define our $RCMSE$ metric as follows:

$$RCMSE = \frac{\widehat{MS}_s - \widehat{MS}_{FOM}}{\widehat{MS}_{FOM}} \tag{4}$$

Both, FOMs and SOMs were generated by using the Judy mutation testing tool for Java. Mutation operators implemented in Judy are presented in detail in Table 3. It is worth mentioning that removing all equivalent mutants first is not feasible if we analyse real world projects, i.e. the large number of mutants, as we did.

Our $RCMSE$ metric is expressed as a ratio and is a unitless number. By multiplying this ratio by 100 it can be expressed as percentage, so the term percentage change in mutations score estimations may also be used.

Table 7 presents results regarding $RCMSE$ and confirms (as we expected) that there is some difference in mutation score estimations between FOM and SOMs.

TABLE 7
Relative change in mutation score estimations

Project name	**FOM**	**SOM strategies:** *Random Mix*	*Last2 First*	*Judy DiffOp*	*Neigh Pair*
Barbecue	0	-.198	-.214	.152	-.055
Commons IO	0	.045	.051	.024	.016
Commons Lang	0	.170	.172	.166	.141
Commons Math	0	.166	.135	.141	.128

TABLE 8
Relative change in mutation score indicators

Project name	**FOM**	**SOM strategies:** *Random Mix*	*Last2 First*	*Judy DiffOp*	*Neigh Pair*
Barbecue	0	.198	.228	.837	.136
Commons IO	0	.077	.085	.059	.017
Commons Lang	0	.235	.240	.236	.169
Commons Math	0	.215	.182	.192	.160

Finding 4: Second order mutation strategies affected mutation score estimations and, as a result, $RCMSE$ defined by Equation 4. In most of the cases mutation score estimations were higher for the SOM strategies than for FOM.

This finding can be seen as a disadvantage of the SOM strategies, as it may suggest that our second order mutants could be easier to kill than first order mutants. Furthermore, the relative change grows over 0.1 in 11 of 16 subjects. Therefore, we will further investigate this issue in Section 6.5.

6.5 Relative change in mutation score indicators

We measured the relative change in mutation score indicators for the SOM strategy (or the HOM strategy in general) in comparison to FOM. We named the metric the relative change in mutation score indicators ($RCMSI$) and defined as follows:

$$RCMSI = \frac{MSI_s - MSI_{FOM}}{MSI_{FOM}} \qquad (5)$$

where MSI_{FOM} is the mutation score indicator (see Equation 2) calculated by means of the classic FOM strategy, while MSI_s is the mutation score indicator calculated by means of the analysed SOM/HOM strategy (s).

Our $RCMSI$ metric is also expressed as a ratio and is a unitless number. By multiplying this ratio by 100 it can be expressed as percentage, so the term percentage change in mutations score indicators may also be used. Summarizing, our $RCMSI$ metric allows us to compare the FOM and the SOM results in mutation score indicators.

Table 8 presents results regarding $RCMSI$ and confirms (as we expected) that there is some difference in mutation score indicators between FOM and SOMs.

Finding 5: Second order mutation strategies affected mutation score indicators and, as a result, $RCMSI$ defined by Equation 5. In all of the cases mutation score indicators were higher for the SOM strategies than for FOM.

The obtained results suggest that unit tests included in the analysed software projects killed proportionally fewer first order mutants than second order mutants in all of the projects and all of the SOM strategies. Furthermore, the relative change grows over 0.1 in 12 of 16 subjects. These results strengthen the conviction from Section 6.4 that our second order mutants appear to be easier to kill then first order mutants. Hence, further research will be focused on obtaining better higher order mutants. A promising way to achieve that goal, suggested by Harman et al. [23], [29], [30], [45], is to search for the HOMs which can subsume their first order counterparts (a subsuming HOM is harder to kill than the FOMs from which it is constructed), thereby reducing test effort without reducing test effectiveness.

6.6 Time of mutation testing process

One of the main reasons why mutation testing is not used in industrial projects is the fact that it is a highly time-consuming process. Fortunately, in our case, the number of generated mutants decreased by 50–72% due to the applied SOM strategies (72% was obtained in Barbecue project when the *JudyDiffOp* SOM strategy was used, see Table 4). As a result, SOM caused a useful reduction in the time needed for testing mutants even though there is some overhead for generating the second order mutants instead of the first order ones. Fortunately, SOM generation time accounts for only about 3% of the total time. The total time spent on the mutation testing process is presented in Table 9.

TABLE 9
Time spent on mutation testing process in seconds.

Project name	**FOM**	**SOM strategies:** *Random Mix*	*Last2 First*	*Judy DiffOp*	*Neigh Pair*
Barbecue	18.83	8.85	9.25	5.95	8.81
Comm. IO	315.32	178.97	231.00	132.27	219.43
Comm. Lang	1207.31	600.86	504.03	359.84	536.17
Comm. Math	1727.16	642.59	688.46	498.75	534.42

The overall time for running the *JudyDiffOp* SOM strategy dropped 67% on average, as compared with first order mutation. It may be noticed that those results are strictly related to the reduction of generated mutants as presented in Table 4, e.g. the average reduction of mutants in the case of the *JudyDiffOp* SOM strategy equals 63.5%.

On the basis of the empirical results (presented in this section) and statistical analysis of the mutation testing time (described in detail in Appendix A4 [51]) one may arrive at the conclusion labelled as Finding 6.

Finding 6: The second order mutation strategy called *JudyDiffOp* significantly reduced the mutation testing time in comparison with first order mutation. The size of the effect was large.

6.7 Manual mutants classification time

The time required for manual assessment (whether the mutant is equivalent or non-equivalent) is well known as one of the vital problems in mutation testing. As observed by Schuler and Zeller [72], it takes on average 14 minutes and 28 seconds to assess one single first order mutation for equivalence. We have measured the classification time for FOMs as well as SOMs, as an extension of Schuler and Zeller's study and, as in their investigation, the variance was high. The minimum classification time for FOM was 2 minutes 5 seconds, while the maximum was 26 minutes 40 seconds. For SOMs, the boundary classification time was 55 seconds and 26 minutes. The obtained results are shown in Table 10. We assessed 200 FOMs (sample size of 50 mutants for each of the four projects) and 800 SOMs (sample size of 50 mutants; four strategies for each of the four projects—all randomly selected).

TABLE 10
FOMs and SOMs classification times [min:sec].

Project name	**FOM**	**SOM strategies:** *Random Mix*	*Last2 First*	*Judy DiffOp*	*Neigh Pair*	*SOM Ave*
Barbecue	11:49	10:13	10:08	09:34	09:59	09:58
Comm. IO	12:56	09:44	08:57	09:26	09:26	09:23
Comm. Lang	11:13	08:16	08:32	08:04	09:42	08:39
Comm. Math	13:10	10:02	10:40	09:34	11:21	10:24
Average time	12:17	09:34	09:34	09:09	10:07	09:36

One can easily see that the average classification time for the SOM strategies is shorter than for FOM. This might be explained as the effect of the second order mutations' characteristics, as described in detail by Polo et al. [65, Table I], e.g. if one of the constituent first order mutations involves examining large parts of the program we can, instead, focus on the second constituent FOM which might be easier to assess. According to Polo et al. [65], the combination of two first-order non-equivalent mutants produces, in general, one second-order non-equivalent mutant. The exception to that rule is possible, but extremely rare. Furthermore, if one of the two first-order mutants is non-equivalent, then the second order mutant is non-equivalent as well (see [65, Table I]). As a result, the time spent on the manual assessment of mutants may be minimized in the case of a second-order mutation.

On the basis of the empirical results (presented in this section) and statistical analysis of manual mutants' classification times (described in detail in Appendix A5 [51]), one may come to the conclusion labelled as Finding 7.

Finding 7: The second-order mutation strategy significantly reduced the time needed for the manual classification of mutants as equivalent or non-equivalent in comparison with the first-order mutation. The size of the effect was medium. A more detailed analysis shows that each of the second-order mutation strategies (i.e. *JudyDiffOp*, *RandomMix*, *Last2First*, *NeighPair*) significantly reduced the time needed for the manual classification of mutants as equivalent or non-equivalent in comparison with the first-order mutation, while the size of the effects were considered small to medium.

6.8 Summary of the experimental results

The experiment indicated strongly that SOM in general and *JudyDiffOp* strategy in particular increase most the efficiency of mutation testing and provide the best results in all but one (the relative change in mutation scores measured via $RCMSE$ and $RCMSI$) of the investigated areas:

1) There was a significant difference in the total number of mutants generated using the FOM and the four SOM strategies ($\chi^2(4) = 16.00$, $p < .001$). Using *JudyDiffOp* SOM strategy (as well as the other analysed SOM strategies) instead of FOM significantly reduced the total number of mutants, while the effect size was large ($r = .65$, $\hat{A} = 1$).
2) There was a significant association between the type of mutation strategy (i.e. FOM vs. SOM) and whether the generated mutant was equivalent or not ($\chi^2(1) = 30.066$, $p < .001$), while the effect size was medium (the odds ratio was 2.57).
3) The number of not killed mutants was significantly affected by mutation strategy applied ($\chi^2(4) = 14.20$, $p < .001$). *JudyDiffOp* SOM strategy significantly reduced the number of not killed mutants in comparison to FOM, while the effect size was large ($r = .65$, $\hat{A} = 1$).
4) The SOM strategies negatively affected mutation scores measured via relative change in mutation score estimations ($RCMSE$) and relative change in mutation score indicators ($RCMSI$), so there is still an area for improvement of the SOM strategies.
5) The mutation testing time was significantly affected by the mutation strategy applied ($\chi^2(4) = 13.60$, $p = .001$). *JudyDiffOp* SOM strategy significantly reduced the mutation testing time in comparison with FOM, while the effect size was large ($r = .65$, $\hat{A} = 1$).
6) Using SOM instead of FOM significantly reduced the time needed for manual classification of mu-

tants as equivalent or non-equivalent ($t(998) = 6.68$, $p < .001$), while the effect size was medium ($r = .21$).

A visual summary of the experimental results related to the number of mutants is presented in Figure 8. The numbers of killed and live mutants were added up across the four analysed projects. The numbers of equivalent and non-equivalent mutants in the analysed projects are estimated on the basis of manually classified samples (1000 mutants were classified manually).

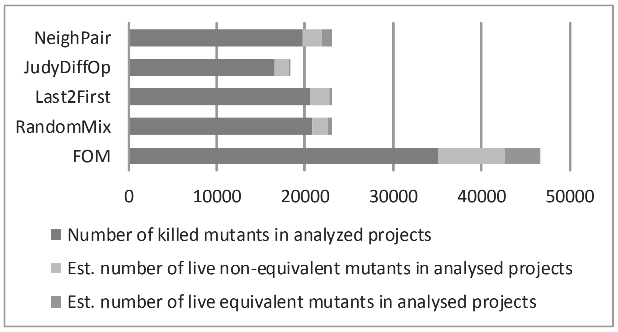


Fig. 8. Comparison of mutation strategies with regard to the number of mutants in four analysed projects

7 Threats to validity

When conducting an experiment, there are always threats to the validity of the results (the validity threats for the systematic literature review have already been discussed separately in Section 3.5). Here, the main threats are discussed on the basis of the list of threats by Cook and Campbell [10] and later described in the context of software engineering by, for example, Wohlin et al. [77] and Madeyski [50].

The *internal* validity of the experiment concerns our ability to draw conclusions about the connections between our independent and dependent variables [77]. There may be threats related to the manual assessment of mutants' equivalence. This part might also be subject to errors and bias. To reduce that threat, the manual cross verification of the obtained results was undertaken between two researchers.

External validity refers to our ability to generalize the results of our study [77]. We examined 50 sample mutations for each strategy and for each of the four non-trivial open source projects. The code size of the analysed projects is larger than in other studies (as shown in Section 5.1).

Even though the analysed projects have disparate characteristics, there is no guarantee that the same results will be obtained for other, very different programs (e.g. with poor code coverage or low fault detection effectiveness measured by the mutation score indicator). However, taking into account the size of the effects and practical implications of the presented results, the relevance to industry, which is a part of external validity [50], seems to be plausible.

Threats to *construct* validity are "the degree to which the independent and the dependent variables are accurately measured by the measurement instrument used in the experiment" [77]. The counting of generated mutants was fully automated in the Judy mutation testing tool. Regarding the manual assessment of mutants' equivalence, the ultimate measure of whether a mutant is non-equivalent is whether or not we are able to write a test which detects a mutation [72]. Preventing possible diffusion or imitation of treatments (i.e. mutation testing strategies) was never an issue since Judy mutation testing tool prevents it.

Threats to the *statistical conclusion* "refers to the appropriate use of statistics to infer whether the presumed independent and dependent variables covary" [10]. To address the risk of low statistical power, we selected a sample size of 50 mutants for each of the four analysed projects and five strategies (1,000 manually classified mutants in total). Moreover, for the sampling method, true random numbers were used. Even though it would have been appropriate to choose a more sophisticated sampling technique with a larger sample size, a researcher has to strike a balance between generalizability and statistical power [3], as well as the effort. Violating the assumptions of statistical tests was minimized by means of non-parametric statistics, as well as the careful checking of the assumptions in cases where parametric tests were used (see Appendix A [51] for further discussion of issues related to statistical tests).

8 Discussion

We interpret our results in such a way that using second order mutant generation strategies, in particular the *JudyDiffOp* algorithm, has a positive influence on effectiveness in solving the equivalent mutant problem. However, an alternative explanation of the results could be that the SOM approach *hides* some of the equivalent mutants behind killable mutants. This is something which needs further research as it may mean that using SOMs to reduce the number of equivalent mutants is not cost effective. Moreover, the authors have discovered that manual classification of second order mutants, against its equivalence, takes less time than with first order mutants.

One contribution of the SLR in comparison to Jia and Harman's survey [31] is what we believe to be a more complete list of existing solutions for the equivalent mutant problem in particular. With a much deeper review of available sources, including coming into contact with all relevant authors and scanning their personal websites, more methods were identified. We have, in particular, investigated the idea of avoiding the equivalent mutant generation techniques as an additional group of approaches and found some omitted methods in other categories. Obviously, because of the focus on the equivalent mutant problem, and having more delimited research questions, our study results are, in our opinion, of high quality from the perspective of EMP. What is important, nevertheless, is that taking a subset of Jia and Harman's results will not give as complete a view on EMP as our SLR.

We have, in addition, proposed the first ever categorization of the existing techniques for EMP. With an increasing number of publications in this field of study, such a classification will, it is our hope, improve transparency and allow for a better understanding of the benefits, disadvantages and differences between methods. Also, because we included theoretical and unproven ideas on how to improve the existing methods and, furthermore, provided what we believe to be a complete review of EMP, this SLR might be a good starting point for future work.

Our contribution from the comparative experiment is the idea, implementation and empirical evaluation of the new as well as the existing strategies for generating second order mutants.

The subsequent contribution of this experiment was the independent investigation of the characteristics of practical application of three existing strategies as proposed by Polo et al. [65]. Polo et al. and Papadakis and Malevris [63] have evaluated these strategies before, but only on small projects. In our research, four larger open source projects were used. Additionally, the authors of the previous studies, in particular [63], [65], used some approximation instead of manual mutants evaluation (as we did).

One additional result of our experiment is the measured time for manual mutant classification against its equivalence. This was the second documented measurement of first order mutants assessment. The first was made by Schuler and Zeller [72] on 140 mutants. In our study, a bigger sample size was used (200 FOMs). Moreover, we are the first who also documented the manual second order mutants classification (800 SOMs), that is, the basis to estimate the real cost of mutation testing (including equivalent mutant elimination).

Our additional contribution which is directly connected with the aforementioned contributions is a tool — Judy mutation testing tool for Java — which is under development with two early versions available online [53]. We believe that our tool may have a positive impact on research and practice in this area.

We believe that the above mentioned contributions make this work important for future mutation testing research. Identifying all methods for EMP, classifying them and collecting the ideas for improvements is by itself valuable, while the investigation of the behaviour of the existing algorithms should be relevant for companies interested in mutation testing.

9 Conclusions and future work

In our opinion, mutation testing is not widely used, mainly because of the problem of efficiency, the generation of too many equivalent mutants, and lack of reliable and usable tools able to integrate with different software development infrastructures and processes. This paper examined a second order mutation approach to deal with those issues specifically. We evaluated the concept of using a set of second order mutants by applying them to large open source software and, thus, increasing the generalizability of this approach. For our experiment we implemented, in the Judy mutation testing tool, different algorithms: *Last2First*, *RandomMix*, *JudyDiffOp*, and *NeighPair*. The first two algorithms were proposed by Polo et al. [65]. The idea for the third one (coined by Polo et al. [65]) was improved by us, while the fourth one was completely new.

This study shows that second order mutation techniques can significantly improve the efficiency of mutation testing at a cost in the testing strength (see Sections 6.4 and 6.5). All four SOM strategies reduced the number of generated mutants by 50% or more. Furthermore, the amount of equivalent mutants has been notably decreased for three of the four strategies. The best results were achieved with the *JudyDiffOp* algorithm. An alternative explanation of the results could be that the SOM approach *hides* some of the equivalent mutants behind killable mutants and it may be subject to future research. What is more, the measured time needed for the classification of equivalent mutants, of both first and second order, indicates quite strongly that the time needed to manually evaluate mutants can be reduced even more when using second order mutation.

The reduction of generated mutants caused a decrease in the time needed for their execution, with approximately 30% of the original time for the most efficient algorithm, i.e. *JudyDiffOp*.

It is also worth noting that second and higher order mutation are not only valuable as a way to address EMP. They also allow us to look at fault masking and to address subtle faults [29]. Previously, this was thought simply impossible because of the large number of mutants required, but it has been shown that search based optimisation algorithms [24] can tame this space quite nicely, so that we can search for good HOMs and need not consider all of them [23], [29], [30].

There is still much work to be done in the field of mutation testing. This paper shows, in our opinion, that second order mutation can be an interesting solution for common problems in mutation testing; however, that can be developed further. Mutants of a higher than second order should be tested on larger programs and other strategies (e.g. employing search based approach) might also be considered. Additionally, the combination of higher order and selective mutation could reduce both the number of equivalent mutants and the execution time even further.

Acknowledgment

The authors would like to express their gratitude to all the researchers who offered their invaluable help in the last phase of the primary studies selection and devoted their efforts to reviewing the preliminary lists of papers.

The authors would also like to thank Paul Ammann, Lydie du Bousquet, Zhenyu Chen, Fabiano Cutigi Ferrari, Mark Harman, Rob Hierons, Garrett Kaminski, Jeff Offutt, Mike Papadakis, Andreas Zeller for their valuable comments on relevant literature and for enabling access to the papers which were otherwise inaccessible.

Finally, we would like to thank the anonymous reviewers for their time and the valuable comments they provided.

References

[1] K. Adamopoulos, M. Harman, and R. M. Hierons, "How to overcome the equivalent mutant problem and achieve tailored selective mutation using co-evolution," in *In GECCO (2), volume 3103 of Lecture Notes in Computer Science*, ser. Lecture Notes in Computer Science, vol. 3103. Springer Berlin / Heidelberg, 2004, pp. 1338–1349.

[2] P. Ammann and J. Offutt, *Introduction to Software Testing.* Cambridge University Press, Cambridge, UK, 2008.

[3] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *ACM/IEEE International Conference on Software Engineering (ICSE).* IEEE, 2011, pp. 1–10.

[4] G. Ausiello, M. Protasi, A. Marchetti-Spaccamela, G. Gambosi, P. Crescenzi, and V. Kann, *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*, 1st ed. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1999.

[5] M. Babar and H. Zhang, "Systematic literature reviews in software engineering: Preliminary results from interviews with researchers," in *Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on*, Oct 2009, pp. 346–355.

[6] D. Baldwin and F. G. Sayward, "Heuristics for determining equivalence of program mutations," Yale University, New Haven, Connecticut, techreport 276, 1979.

[7] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, "Lessons from applying the systematic literature review process within the software engineering domain," *Journal of Systems and Software*, vol. 80, no. 4, pp. 571–583, 2007.

[8] T. A. Budd and D. Angluin, "Two notions of correctness and their relation to testing," *Acta Informatica*, vol. 18, pp. 31–45, 1982.

[9] T. A. Budd, R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "The design of a prototype mutation system for program testing," in *Proceedings of the AFIPS National Computer Conference*, Anaheim, New Jersey, Jun 1978, pp. 623–627.

[10] T. D. Cook and D. T. Campbell, *Quasi-Experimentation: Design and Analysis Issues for Field Settings.* Houghton Mifflin, 1979.

[11] M. E. Delamaro and J. C. Maldonado, "Proteum - A tool for the assessment of test adequacy for C programs," in *Proceedings of the Conference on Performability in Computing Systems (PCS'96)*, New Brunswick, New Jersey, Jul 1996, pp. 79–95.

[12] R. DeMillo, R. Lipton, and F. Sayward, "Hints on test data selection: Help for the practicing programmer," *Computer*, vol. 11, no. 4, pp. 34–41, Apr 1978.

[13] A. Derezińska and A. Szustek, "CREAM - A system for object-oriented mutation of C# programs," Annals Gdańsk University of Technology Faculty ETI, No 5, Information Technologies vol.13, Gdańsk 2007, pp. 389-406, ISBN 978-83-60779-01-9

[14] A. Derezińska and A. Szustek, "Tool-supported advanced mutation approach for verification of C# programs," in *International Conference on Dependability of Computer Systems, 2008. DepCos-RELCOMEX '08.* Los Alamitos, CA, USA: IEEE Computer Society, june 2008, pp. 261–268.

[15] L. du Bousquet and M. Delaunay, "Towards mutation analysis for Lustre programs," *Electronic Notes in Theoretical Computer Science*, vol. 203, no. 4, pp. 35–48, 2008.

[16] S. Eldh, S. Punnekkat, H. Hansson, and P. JÃűnsson, "Component testing is not enough - a study of software faults in telecom middleware," in *Testing of Software and Communicating Systems*, ser. Lecture Notes in Computer Science, A. Petrenko, M. Veanes, J. Tretmans, and W. Grieskamp, Eds. Springer Berlin Heidelberg, 2007, vol. 4581, pp. 74–89. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-73066-8_6

[17] M. Ellims, D. Ince, and M. Petre, "The Csaw C mutation tool: Initial results," in *Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION.* Washington, DC, USA: IEEE Computer Society, 2007, pp. 185–192. [Online]. Available: http://portal.acm.org/citation.cfm?id=1308173.1308282

[18] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness.* New York, NY, USA: W. H. Freeman & Co., 1990.

[19] L. Goodman, "Snowball sampling," *The Annals of Mathematical Statistics*, vol. 32, no. 1, pp. 148–170, Mar 1961.

[20] B. J. M. Grün, D. Schuler, and A. Zeller, "The impact of equivalent mutants," in *Proceedings of the IEEE International Conference on Software Testing, Verification, and Validation Workshops.* Denver, Colorado, USA: IEEE Computer Society, 2009, pp. 192–199.

[21] R. Hamlet, "Testing programs with the aid of a compiler," *IEEE Transactions on Software Engineering*, vol. SE-3, no. 4, pp. 279–290, Jul 1977.

[22] M. Harman, R. Hierons, and S. Danicic, "The relationship between program dependence and mutation analysis," in *Mutation testing for the new century*, W. E. Wong, Ed. Norwell, MA, USA: Kluwer Academic Publishers, 2001, pp. 5–13. [Online]. Available: http://portal.acm.org/citation.cfm?id=571305.571310

[23] M. Harman, Y. Jia, and W. B. Langdon, "A manifesto for higher order mutation testing," in *Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops*, ser. ICSTW '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 80–89.

[24] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Comput. Surv.*, vol. 45, no. 1, pp. 11:1–11:61, Dec. 2012. [Online]. Available: http://doi.acm.org/10.1145/2379776.2379787

[25] R. Hierons, M. Harman, and S. Danicic, "Using program slicing to assist in the detection of equivalent mutants," *Software Testing, Verification and Reliability*, vol. 9, no. 4, pp. 233–262, 1999.

[26] E. Hossain, M. A. Babar, and H.-y. Paik, "Using Scrum in global software development: A systematic literature review," in *Proceedings of the 2009 Fourth IEEE International Conference on Global Software Engineering*, ser. ICGSE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 175–184.

[27] 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," in *Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION.* Washington, DC, USA: IEEE Computer Society, 2007, pp. 169–175. [Online]. Available: http://portal.acm.org/citation.cfm?id=1308173.1308280

[28] C. Ji, Z. Chen, B. Xu, and Z. Wang, "A new mutation analysis method for testing Java exception handling," in *Proc. 33rd Annual IEEE Int. Computer Software and Applications Conf. COMPSAC '09*, vol. 2, 2009, pp. 556–561.

[29] Y. Jia and M. Harman, "Constructing subtle faults using higher order mutation testing," in *Source Code Analysis and Manipulation*, 2008, pp. 249–258.

[30] Y. Jia and M. Harman, "Higher order mutation testing," *Information and Software Technology*, vol. 51, pp. 1379–1393, Oct 2009.

[31] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Transactions on Software Engineering*, vol. 37, no. 5, pp. 649–678, 2011, (available earlier as a technical report http://www.dcs.kcl.ac.uk/pg/jiayue/repository/TR-09-06.pdf).

[32] Y. Jia and M. Harman, "Mutation Testing Repository," 2012. [Online]. Available: http://crestweb.cs.ucl.ac.uk/resources/mutation_testing_repository/index.php

[33] G. Kaminski and P. Ammann, "Using a fault hierarchy to improve the efficiency of DNF logic mutation testing," in *Proc. Int. Conf. Software Testing Verification and Validation ICST '09*, 2009, pp. 386–395.

[34] K. S. Khan, G. T. Riet, J. Glanville, A. J. Sowden, and J. Kleijnen, "Undertaking systematic reviews of research on

effectiveness CRD," University of York, Tech. Rep. 4, 2001. [Online]. Available: http://opensigle.inist.fr/handle/10068/534964

[35] S. U. Khan, M. Niazi, and R. Ahmad, "Barriers in the selection of offshore software development outsourcing vendors: An exploratory study using a systematic literature review," *Information and Software Technology*, vol. 53, no. 7, pp. 693 – 706, 2011.

[36] K. N. King and A. J. Offutt, "A Fortran language system for mutation-based software testing," *Software - Practice and Experience*, vol. 21, pp. 685–718, Jun 1991.

[37] M. Kintis, M. Papadakis, and N. Malevris, "Evaluating mutation testing alternatives: A collateral experiment," in *Proc. 17th Asia Pacific Software Engineering Conf. (APSEC)*, 2010, pp. 300–309.

[38] B. Kitchenham, "Procedures for performing systematic reviews," Keele University and NICTA, Tech. Rep., 2004.

[39] B. Kitchenham, P. Brereton, M. Turner, M. Niazi, S. Linkman, R. Pretorius, and D. Budgen, "The impact of limited search procedures for systematic literature reviews – A participant-observer case study," in *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 336–345.

[40] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Keele University and Durham University Joint Report, Tech. Rep. EBSE 2007-001, 2007.

[41] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering - A systematic literature review," *Information and Software Technology*, vol. 51, no. 1, pp. 7–15, 2008.

[42] B. Kitchenham, R. Pretorius, D. Budgen, O. Pearl Brereton, M. Turner, M. Niazi, and S. Linkman, "Systematic literature reviews in software engineering: A tertiary study," *Information and Software Technology*, vol. 52, pp. 792–805, Aug 2010.

[43] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam, and J. Rosenberg, "Preliminary guidelines for empirical research in software engineering," *IEEE Transactions on Software Engineering*, vol. 28, no. 8, pp. 721–734, 2002.

[44] W. B. Langdon, M. Harman, and Y. Jia, "Multi objective higher order mutation testing with genetic programming," in *Proceedings of the 2009 Testing: Academic and Industrial Conference - Practice and Research Techniques*, ser. TAIC-PART '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 21–29.

[45] W. B. Langdon, M. Harman, and Y. Jia, "Efficient multi-objective higher order mutation testing with genetic programming," *Journal of Systems and Software*, vol. 83, no. 12, pp. 2416–2430, Dec. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2010.07.027

[46] Y.-S. Ma, J. Offutt, and Y. R. Kwon, "MuJava: An automated class mutation system," *Software Testing, Verification and Reliability*, vol. 15, no. 2, pp. 97–133, 2005.

[47] L. Madeyski, "On the effects of pair programming on thoroughness and fault-finding effectiveness of unit tests," in *Product-Focused Software Process Improvement*, ser. Lecture Notes in Computer Science, J. Münch and P. Abrahamsson, Eds. Springer Berlin / Heidelberg, 2007, vol. 4589, pp. 207–221. [Online]. Available: http://madeyski.e-informatyka.pl/download/Madeyski07.pdf

[48] L. Madeyski, "Impact of pair programming on thoroughness and fault detection effectiveness of unit test suites," *Software Process: Improvement and Practice*, vol. 13, no. 3, pp. 281–295, 2008. [Online]. Available: http://madeyski.e-informatyka.pl/download/Madeyski08.pdf

[49] L. Madeyski, "The impact of test-first programming on branch coverage and mutation score indicator of unit tests: An experiment," *Information and Software Technology*, vol. 52, pp. 169–184, February 2010. [Online]. Available: http://madeyski.e-informatyka.pl/download/Madeyski10c.pdf

[50] L. Madeyski, *Test-Driven Development: An Empirical Evaluation of Agile Practice.* (Heidelberg, London, New York): Springer, 2010, Foreword by Prof. Claes Wohlin. [Online]. Available: http://www.springer.com/978-3-642-04287-4

[51] L. Madeyski, W. Orzeszyna, R. Torkar, and M. Józala. (2012) Appendix to the paper "Overcoming the Equivalent Mutant Problem: A Systematic Literature Review and a Comparative Experiment of Second Order Mutation". [Online]. Available: http://madeyski.e-informatyka.pl/download/app/AppendixTSE.pdf

[52] L. Madeyski and N. Radyk, "Judy – A mutation testing tool for Java," *IET Software*, vol. 4, no. 1, pp. 32–42, Feb 2010. [Online]. Available: http://madeyski.e-informatyka.pl/download/Madeyski10b.pdf

[53] "Judy: A mutation testing tool for Java," 2012. [Online]. Available: http://madeyski.e-informatyka.pl/tools/judy/

[54] E. Martin and T. Xie, "A fault model and mutation testing of access control policies," in *Proceedings of the 16th international conference on World Wide Web*, ser. WWW '07. New York, New York, USA: ACM Press, 2007, pp. 667–676.

[55] E. S. Mresa and L. Bottaci, "Efficiency of mutation operators and selective mutation strategies: An empirical study," *Software Testing, Verification and Reliability*, vol. 9, no. 4, pp. 205–232, 1999.

[56] A. J. Offutt and J. Pan, "Detecting equivalent mutants and the feasible path problem," in *Proc. Eleventh Annual Conf. 'Systems Integrity Computer Assurance COMPASS '96 Software Safety. Process Security'*, 1996, pp. 224–236.

[57] A. J. Offutt, "Investigations of the software testing coupling effect," *ACM Transactions on Software Engineering Methodology*, vol. 1, pp. 5–20, Jan 1992.

[58] A. J. Offutt and W. M. Craft, "Using compiler optimization techniques to detect equivalent mutants," *Software Testing, Verification and Reliability*, vol. 4, no. 3, pp. 131–154, 1994.

[59] A. J. Offutt and J. Pan, "Automatically detecting equivalent mutants and infeasible paths," *Software Testing, Verification and Reliability*, vol. 7, no. 3, pp. 165–192, 1997.

[60] A. J. Offutt and R. H. Untch, "Mutation 2000: Uniting the orthogonal," in *Mutation testing for the new century*, W. E. Wong, Ed. Norwell, MA, USA: Kluwer Academic Publishers, 2001, pp. 34–44. [Online]. Available: http://portal.acm.org/citation.cfm?id=571305.571314

[61] J. Offutt, Y.-S. Ma, and Y.-R. Kwon, "The class-level mutants of MuJava," in *Proceedings of the 2006 international workshop on Automation of software test - AST '06*, ser. AST '06. New York, New York, USA: ACM Press, 2006, pp. 78–84.

[62] W. Orzeszyna, L. Madeyski, and R. Torkar. Protocol for a systematic literature review of methods dealing with equivalent mutant problem. Wroclaw University of Technology. [Online]. Available: http://madeyski.e-informatyka.pl/download/slr/EquivalentMutantsSLRProtocol.pdf

[63] M. Papadakis and N. Malevris, "An empirical evaluation of the first and second order mutation testing strategies," in *Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops*, ser. ICSTW '10. IEEE Computer Society, 2010, pp. 90–99.

[64] C. H. Papadimitriou, *Computational complexity.* Addison-Wesley, 1994.

[65] M. Polo, M. Piattini, and I. García-Rodríguez, "Decreasing the cost of mutation testing with second-order mutants," *Softw. Test. Verif. Reliab.*, vol. 19, pp. 111–131, June 2009.

[66] R. Purushothaman and D. E. Perry, "Toward understanding the rhetoric of small source code changes," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 511–526, Jun. 2005. [Online]. Available: http://dx.doi.org/10.1109/TSE.2005.74

[67] J. A. Rosenthal, "Qualitative descriptors of strength of association and effect size," *Journal of Social Service Research*, vol. 21, no. 4, pp. 37–59, 1996.

[68] J. Schöpfel, "Towards a prague definition of grey literature," in *Twelfth International Conference on Grey Literature: Transparency in Grey Literature, Grey Tech Approaches to High Tech Issues, 6-7 December 2010*, 2010. [Online]. Available: http://archivesic.ccsd.cnrs.fr/docs/00/58/15/70/PDF/GL_12_Schopfel_v5.2.pdf

[69] D. Schuler and A. Zeller, "(Un-)covering equivalent mutants," in *Proceedings of the 3rd International Conference on Software Testing Verification and Validation (ICST'10)*, Paris, France, Apr 2010, pp. 45–54.

[70] D. Schuler, V. Dallmeier, and A. Zeller, "Efficient mutation testing by checking invariant violations," in *ISSTA '09: Proceedings of the Eighteenth International Symposium on Software Testing and Analysis*, ser. ISSTA '09. New York, New York, USA: ACM Press, 2009, pp. 69–80.

[71] D. Schuler and A. Zeller, "Javalanche: Efficient mutation testing for Java," in *ESEC/FSE '09: Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. ESEC/FSE '09. New York, New York, USA: ACM Press, 2009, p. 297.

[72] D. Schuler and A. Zeller, "Covering and uncovering equivalent mutants," *Software Testing, Verification and Reliability*, pp. n/a–n/a, 2012. [Online]. Available: http://dx.doi.org/10.1002/stvr.1473

[73] B. H. Smith and L. Williams, "An Empirical Evaluation of the MuJava Mutation Operators," *Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION 2007*, pp. 193–202, 10-14 Sept. 2007.

[74] "Jumble: A class level mutation testing tool," SourceForge, April 2011. [Online]. Available: http://jumble.sourceforge.net

[75] F. Steimann and A. Thies, "From behaviour preservation to behaviour modification: Constraint-based mutant generation," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 425–434.

[76] A. M. R. Vincenzi, E. Y. Nakagawa, J. C. Maldonado, M. E. Delamaro, and R. A. F. Romero, "Bayesian-learning based guidelines to determine equivalent mutants," *International Journal of Software Engineering and Knowledge Engineering*, vol. 12, no. 6, pp. 675–690, 2002.

[77] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering: An introduction*. Norwell, MA, USA: Kluwer Academic Publishers, 2000.

[78] W. E. Wong, Ed., *Mutation testing for the new century*. Norwell, MA, USA: Kluwer Academic Publishers, 2001.

[79] H. Zhu, P. A. V. Hall, and J. H. R. May, "Software unit test coverage and adequacy," *ACM Computing Surveys*, vol. 29, pp. 366–427, Dec 1997.

Lech Madeyski received the PhD and Habilitation (DSc) degrees in computer science from the Wroclaw University of Technology, Poland, in 1999 and 2011 respectively. He is currently an Assistant Professor at Wroclaw University of Technology, Poland. His research focus is on empirical, quantitative research methods and machine learning in the field of software engineering. He is one of the founders of the International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE) series which started in 2006 in Erfurt. He is the author of a book on empirical evaluation (via statistical analyses and meta-analyses) of Test-Driven Development agile software development practice, published by Springer in 2010. He is a member of ACM and IEEE.

Wojciech Orzeszyna received his MSc degrees in software engineering from Wroclaw University of Technology, Poland (supervised by Lech Madeyski) and Blekinge Institute of Technology, Sweden (supervised by Richard Torkar) in 2011. He is a software developer working in industry. His interests include software craftsmanship, quality and agile practices.

Richard Torkar is an Associate Professor at Blekinge Institute of Technology, and Chalmers University of Technology | University of Gothenburg, Sweden. His focus is on quantitative research methods in the field of software engineering. He received his Ph.D. in software engineering from Blekinge Institute of Technology, Sweden, in 2006. He's a member of ACM and IEEE.

Mariusz Józala is a Software Developer. In 2011 he graduated from University of Technology in Wroclaw with MSc degree in Software Engineering (supervised by Lech Madeyski). He has earned his experience in Nokia Solutions and Networks and OpenBet companies. He is an advocate for clean and efficient coding and agile software development practices.