Higher order mutation testing to drive development of new test cases: an empirical comparison of three strategies

Quang Vu Nguyen, Lech Madeyski

{Quang.vu.nguyen; Lech.Madeyski}@pwr.edu.pl
Faculty of Computer Science and Management, Wroclaw University of Technology,
Wybrzeze Wyspianskiego 27, 50-370 Wroclaw, Poland

Abstract. Mutation testing, which includes first order mutation (FOM) testing and higher order mutation (HOM) testing, appeared as a powerful and effective technique to evaluate the quality of test suites. The live mutants, which cannot be killed by the given test suite, make up a significant part of generated mutants and may drive the development of new test cases. Generating live higher order mutants (HOMs) able to drive development of new test cases is considered in this paper. We apply multi-objective optimization algorithms based on our proposed objectives and fitness functions to generate higher order mutants using three strategies: HOMT1 (HOMs generated from all first order mutants), HOMT2 (HOMs generated from killed first order mutants) and HOMT3 (HOMs generated from not-easy-to-kill first order mutants). We then use mutation score indicator to evaluate, which of the three approaches is better suited to drive development of new test cases and, as a result, to improve the software quality.

Keywords. Mutation testing; Higher Order Mutation testing; Live mutants; Equivalent mutants; Multi-objective optimization algorithm.

1 Introduction

In 1970s, a fault-based technique was introduced by DeMillo et al. [1] and Hamlet [2] as a way to measure the effectiveness of test suites, called mutation testing (first order mutation testing). Mutants are the different versions of an original program generated by inserting, via a mutation operator, only one semantic change (or fault) into the original program. Mutation operators depend on programming languages, but there are some traditionally used mutation operators, e.g., deletion of a statement, replacement of Boolean expressions, replacement of arithmetic, replacement of a variable. Given set of test cases (TCs) is executed on the original program and all its mutants.

Quang Vu Nguyen and Lech Madeyski, Intelligent Information and Database Systems: 8th Asian Conference, ACIIDS 2016, Da Nang, Vietnam, March 14-16, 2016, Proceedings, Part I, vol. 9621 of Lecture Notes in Artificial Intelligence, ch. Higher Order Mutation Testing to Drive Development of New Test Cases: An Empirical Comparison of Three Strategies, pp. 235–244. Springer, 2016. DOI: 10.1007/978-3-662-49381-6_23 (URL: http://dx.doi.org/10.1007/978-3-662-49381-6_23, Draft: http://madeyski.e-informatyka.pl/download/NguyenMadeyski16.pdf).

If output result of mutant is different than the output result of original program, with any test case (TC), we say that the mutant is killed. In other words, the test case kills mutant. If a mutant was killed by all of given TCs, it is named "Easy to kill".

Conversely, if output results of mutant and original program are the same with all test cases, the mutant is called "live" or "not killed". In this case, none of the test cases in the given set of test cases can kill the mutant. This could be for two reasons: (1) The given set of test cases is "not good enough" to detect the difference between the original program and its mutants; it drives developers to create new test cases able to kill live mutants. (2) The mutant is an equivalent mutant; it means that the mutant has the same semantic meaning as the original program and there is no test case able to kill the mutant.

The equivalent mutant problem (EMP) is one of the crucial problems in mutation testing [4, 7, 12]. This is one of the reasons why mutation testing is not yet widely adopted in practice. A lot of approaches have been proposed for overcoming the EMP (and the mutation testing's problems in general) including second order mutation testing [7, 8, 9, 10, 11] or higher order mutation testing [3, 4, 5, 6] in general. Higher order mutation testing is an idea presented by Jia and Harman [3] and in a manifesto by Harman et al. [4]. This promising idea offers solutions to overcome the limitations of traditional mutation testing. Mutants can be classified into two types: First Order Mutants (FOMs) and Higher Order Mutants (HOMs). The first are used in traditional mutation testing and generated by applying mutation operators only once in each mutant. The second are used in higher order mutation testing and constructed by inserting two or more changes per mutant.

Mutation score (or mutation adequacy) was defined as the ratio of the number of killed mutants to the number of non-equivalent mutants [13]. The number or non-equivalent mutants is a difference between total number of generated mutants and number of equivalent mutants.

Mutation score indicator (MSI) is another quantitative measure of the quality of test cases. Different from MS, MSI was defined as the ratio of killed mutants to all generated mutants [7, 14, 15, 16, 17]. MSI lies between 0 and 1. If MSI is 0, all generated mutants are live mutants. If MSI is 1, all mutants are killed. Ignoring equivalent mutants means that we accept the lower bound on mutation score. In addition, in fact many mutation operators can produce equivalent mutants of the same behaviour as the original program, while detection of equivalent mutants often involves additional human effort.

Live mutant problem includes equivalent mutants and non-equivalent mutants, which could be killed by adding high quality TCs. So, existing live mutants can drive development of new high quality TCs. Development of new high quality TCs decreases the number of live mutants due to new TCs able to kill some non-equivalent live mutants. New high quality TCs have a positive impact on software quality. Our goal is to investigate which strategy to generate HOMs gives more opportunities to

Quang Vu Nguyen and Lech Madeyski, Intelligent Information and Database Systems: 8th Asian Conference, ACIIDS 2016, Da Nang, Vietnam, March 14-16, 2016, Proceedings, Part I, vol. 9621 of Lecture Notes in Artificial Intelligence, ch. Higher Order Mutation Testing to Drive Development of New Test Cases: An Empirical Comparison of Three Strategies, pp. 235–244. Springer, 2016. DOI: 10.1007/978-3-662-49381-6_23 (URL: http://dx.doi.org/10.1007/978-3-662-49381-6_23, Draft: http://madeyski.e-informatyka.pl/download/NguyenMadeyski16.pdf).

drive development of high quality TCs. Three considered strategies are: 1) HOMT1 - HOMs generated from all first order mutants, 2) HOMT2 - HOMs generated from killed first order mutants and 3) HOMT3 - HOMs generated from not-easy-to-kill first order mutants.

In this paper, we apply three multi-objective optimization algorithms – NSGAII, NSGAIII and eMOEA (Epsilon-MOEA) – to generate HOMs based on our objectives and fitness functions. We use mutation score indicator (MSI) as the indicator of usefulness of higher order mutation in driving development of TCs. Furthermore, HOMs simulate faults, which require more than one change to correct them. This kind of faults, represented by HOMs, is even more realistic than faults represented by FOMs. For example, Purushothaman and Perry [23] found that there is less than 4 percent probability that a one-line change will introduce a fault in the code. Hence, HOMs complement FOMs and enhance realism of mutation testing giving opportunity to simulate more realistic faults and create test cases able to spot these kind of faults. Also the number of equivalent mutants in each of the strategies (FOMT, HOMT1, HOMT2 and HOMT3) would be different. Hence, our dependent variable (MSI) is indeed an approximate measure of how useful each of the strategies can be in driving TCs development.

The rest of the paper is organized as follows. Section 2 includes our objectives and fitness function, which are applied to multi-objective optimization algorithms. Section 3 presents the experimental procedure, the proposed multi-objective optimization algorithms and real-world projects under test. Section 4 shows results of the empirical evaluation. Section 5 discusses threats to validity, while the last section presents conclusions and proposition of future works.

2 Objectives and fitness functions

Based on the idea "number of test cases (TCs) which can kill HOMs is as small as possible", we have proposed objectives and fitness functions [29], which we will apply in the different multi-objective optimization algorithms to generate HOMs. Some notations are explained below (See Figure 1):

H: a HOM, constructed from FOMs: F_1 and F_2

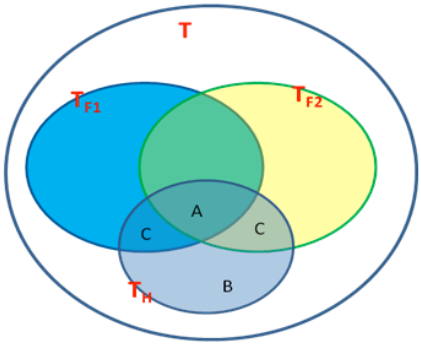
T: The given set of test cases

$T_{F1} \subset T$: Set of test cases that kill FOM1

$T_{F2} \subset T$: Set of test cases that kill FOM2

$T_H \subset T$: Set of test cases that kill H

Quang Vu Nguyen and Lech Madeyski, Intelligent Information and Database Systems: 8th Asian Conference, ACIIDS 2016, Da Nang, Vietnam, March 14-16, 2016, Proceedings, Part I, vol. 9621 of Lecture Notes in Artificial Intelligence, ch. Higher Order Mutation Testing to Drive Development of New Test Cases: An Empirical Comparison of Three Strategies, pp. 235–244. Springer, 2016. DOI: 10.1007/978-3-662-49381-6_23 (URL: http://dx.doi.org/10.1007/978-3-662-49381-6_23, Draft: http://madeyski.e-informatyka.pl/download/NguyenMadeyski16.pdf).



$$A = T_H \cap T_{F1} \cap T_{F2}$$

$$B = T_H \setminus (T_{F1} \cup T_{F2})$$

$$C = (T_H \cap (T_{F1} \cup T_{F2})) \setminus (T_{F1} \cap T_{F2})$$

Fig. 1. The combination of sets of TCs

$A \subset T_H$: Set of test cases that can kill H and all its constituent FOMs.

$B \subset T_H$: Set of test cases that kill H but cannot kill any its constituent FOMs.

$C \subset T_H$: Set of test cases that kill H and can kill FOM1 or FOM2.

The Figure 1 showed that the set of TCs that kills a HOM can be divided into 3 subsets:

- The first is the subset that can kill HOM and all its constituent FOMs (subset A)
- The second is the subset that kills HOM and can kill FOM1 or FOM2 (subset C)
- The third is the subset that only kills HOM and cannot kill any FOMs (subset B)

From that, we have proposed objectives and their fitness functions [29] (see Equations below) to apply multi-objective optimization algorithms to construct HOMs as follows:

Objective 1: Minimize the number of TCs that kill HOM and also kill all its constituent FOMs (The fitness function is fitness(OB1) in Equation 1).

Objective 2: Minimize the number of TCs that kill HOM but cannot kill any their constituent FOMs (The fitness function is fitness(OB2) in Equation 2).

Objective 3: Minimize the number of TCs that kill HOM and can kill FOM1 or FOM2 (The fitness function is fitness(OB3) in Equation 3).

$$fitness(OB1) = \frac{\#(T_H \cap T_{F1} \cap T_{F2})}{\#T_H} \quad (1)$$

$$fitness(OB2) = \frac{\#(T_H \setminus (T_{F1} \cup T_{F2}))}{\#T_H} \quad (2)$$

Quang Vu Nguyen and Lech Madeyski, Intelligent Information and Database Systems: 8th Asian Conference, ACIIDS 2016, Da Nang, Vietnam, March 14-16, 2016, Proceedings, Part I, vol. 9621 of Lecture Notes in Artificial Intelligence, ch. Higher Order Mutation Testing to Drive Development of New Test Cases: An Empirical Comparison of Three Strategies, pp. 235–244. Springer, 2016. DOI: 10.1007/978-3-662-49381-6_23 (URL: http://dx.doi.org/10.1007/978-3-662-49381-6_23, Draft: http://madeyski.e-informatyka.pl/download/NguyenMadeyski16.pdf).

$$fitness(OB3) = \frac{\#((T_H \cap (T_{F1} \cup T_{F2})) \setminus (T_{F1} \cap T_{F2}))}{\#T_H} \quad (3)$$

$$fitness(H) = \frac{\#T_H}{\#(T_{F1} \cup T_{F2})} \quad (4)$$

The values of fitness(OB1), fitness(OB2) and fitness(OB3) lie between 0 and 1. In addition, we also have proposed the fitness(H) function (Equation 4) which is used to evaluate a HOM whether it is harder to kill than its constituent FOMs or not. If the number of TCs that can kill HOM is smaller than the number of TCs that can kill its FOMs, HOM is called harder to kill than its constituent FOMs.

3 Experiment planning and execution

The aim of our experiment was to answer the research question: How to combine FOMs to create hard to kill HOMs (well suited to evaluate the quality of test cases and drive their development)?

3.1 Supporting tool

We use Judy tool [17, 7] to conduct the empirical studies. Judy (http://www.mutationtesting.org/) is a mutation testing tool for Java programs. It supports large set of mutation operators, as well as HOM generation, HOM execution and mutation analysis.

3.2 Multi-objective optimization algorithms

NSGA-II is the second version of the Non-dominated Sorting Genetic Algorithm that was proposed by Deb et al. [18] for solving non-convex and non-smooth single and multi-objective optimization problems. Its main features are: it uses an elitist principle; it emphasizes non-dominated solutions; and it uses an explicit diversity preserving mechanism. NSGA-III is the extension of NSGA-II which is based on the supply of a set of reference points and demonstrated its working in 3 to 15-objective optimization problems [19]. The εMOEA (eMOEA) is a steady state multi-objective evolutionary algorithm that co-evolves both an evolutionary algorithm population and an archive population by randomly mating individuals from the population and the archive to generate new solutions [20, 21].

Quang Vu Nguyen and Lech Madeyski, Intelligent Information and Database Systems: 8th Asian Conference, ACIIDS 2016, Da Nang, Vietnam, March 14-16, 2016, Proceedings, Part I, vol. 9621 of Lecture Notes in Artificial Intelligence, ch. Higher Order Mutation Testing to Drive Development of New Test Cases: An Empirical Comparison of Three Strategies, pp. 235–244. Springer, 2016. DOI: 10.1007/978-3-662-49381-6_23 (URL: http://dx.doi.org/10.1007/978-3-662-49381-6_23, Draft: http://madeyski.e-informatyka.pl/download/NguyenMadeyski16.pdf).

3.3 Projects under test (PUT)

In our empirical study we use five real-world, open source projects (see Table 1) which were downloaded from the SourceForge website (http://sourceforge.net). Table 1 shows the projects selected for the experiment along with their number of classes (NOC), lines of code (LOC) and number of given test cases (#TCs).

Table 1. Projects under test

Project	NOC	LOC	#TCs
BeanBin	72	5925	68
Barbecue	57	23996	190
JWBF	51	13572	305
CommonsChain 1.2	103	13410	17
CommonsValidiator 1.4.1	144	25422	66

3.4 Experimental procedure

For each project under test, we ran the process, which was described in following experimental procedure, 5 times. HOMs were generated in three ways. Firstly, HOMs were created by combining FOMs from the set of all generated FOMs. And second one, delete first live FOMs from set of generated FOMs, then create HOMs by combining FOMs from the set of killed FOMs. And the last, first delete all of easy to kill FOMs, which were killed by all of given TCs, from set of generated FOMs, then create HOMs by combining FOMs from the set of not-easy-to-kill FOMs Then we calculated the average value of each program for each algorithm. We set out the experimental procedure as follows:

```
for each software under test do
Generate all possible FOMs by applying the set of Judy
mutation operators
Count and save MSI of first order mutation testing
Set objectives and fitness functions
  for each multi-objective optimization algorithm do
    - set populationSize =100
    - set maxMutationOrder =15
    - from set of all FOMs, generate and evaluate HOMs,
guided by objectives and fitness functions
    - count and save MSI of higher order mutation testing
    - delete the live FOMs from set of all generated FOMs
```

Quang Vu Nguyen and Lech Madeyski, Intelligent Information and Database Systems: 8th Asian Conference, ACIIDS 2016, Da Nang, Vietnam, March 14-16, 2016, Proceedings, Part I, vol. 9621 of Lecture Notes in Artificial Intelligence, ch. Higher Order Mutation Testing to Drive Development of New Test Cases: An Empirical Comparison of Three Strategies, pp. 235–244. Springer, 2016. DOI: 10.1007/978-3-662-49381-6_23 (URL: http://dx.doi.org/10.1007/978-3-662-49381-6_23, Draft: http://madeyski.e-informatyka.pl/download/NguyenMadeyski16.pdf).

```
    - from set of remaining-FOMs, generate and evaluate
HOMs, guided by objectives and fitness functions
    - count and save MSI of higher order mutation testing
    - delete the easy-to-kill FOMs from set of all gener-
ated FOMs
    - from set of remaining-FOMs, generate and evaluate
HOMs, guided by objectives and fitness functions
    - count and save MSI of higher order mutation testing

  end
end
```

4 Results and analysis

Results were shown in Table 2. FOMT is implementation of first order mutation testing. HOMT1 is the implementation of higher order mutation testing where HOMs are generated on a basis of all FOMs. HOMT2 is the implementation of higher order mutation testing where HOMs are generated on a basis of killed FOMs. HOMT3 is the implementation of higher order mutation testing where HOMs are generated on a basis of not-easy-to-kill FOMs.

Table 2. The mean value of MSI for each project under test (%)

Strategy \ Project Under Test (PUT)		Barbecue	BeanBin	Commons Chain	Commons Validator	JWBF
FOMT		15.79	15.11	42.65	47.10	12.96
HOMT 1	NSGAII	70.59	36.32	89.92	92.41	94.54
	NSGAIII	69.67	43.04	84.38	92.31	91.30
	eMOEA	69.19	41.04	87.55	93.15	91.20
HOMT 2	NSGAII	100	100	100	100	100
	NSGAIII	100	100	100	100	100
	eMOEA	96.15	100	100	99.31	100
HOMT 3	NSGAII	64.01	62.11	40.12	83.93	77.78
	NSGAIII	54.23	59.23	50.28	86.05	80.00
	eMOEA	73.59	69.42	49.67	87.76	90.91

The results presented in Table 2 indicate that the given sets of TCs of PUTs have lower MSI in first order mutation testing. It means that there are many live FOMs and

Quang Vu Nguyen and Lech Madeyski, Intelligent Information and Database Systems: 8th Asian Conference, ACIIDS 2016, Da Nang, Vietnam, March 14-16, 2016, Proceedings, Part I, vol. 9621 of Lecture Notes in Artificial Intelligence, ch. Higher Order Mutation Testing to Drive Development of New Test Cases: An Empirical Comparison of Three Strategies, pp. 235–244. Springer, 2016. DOI: 10.1007/978-3-662-49381-6_23 (URL: http://dx.doi.org/10.1007/978-3-662-49381-6_23, Draft: http://madeyski.e-informatyka.pl/download/NguyenMadeyski16.pdf).

the given sets of TCs are not good enough to detect the difference between original program and their mutants and, therefore, need to be improved following the results of mutation analysis based on the FOMT strategy. The numbers of live FOMs makes up from 52% to 87% of generated mutants. Only a small number of FOMs were killed by the given sets of TCs. In the case of live FOMs, we have to check whether the live FOMs are equivalent mutants or not, but it often involves additional human effort. If mutants are not equivalent, developers or testers create new TCs and check whether they are able to kill live FOMs. If live FOMs are equivalent mutants, TCs, which can kill them, do not exist.

The most striking result is that the HOMT2 strategy appeared to be useless as it gives a false impression that TCs are of high quality (MSI is equal or close to 100%) and the usefulness of HOMT2 is strongly limited, i.e., opportunities of test case improvement guided by results of HOMT2 mutation analysis are rare if any. Almost all of higher degree mutants, which were constructed by combining the killed FOMs, are also killed. This indicates that, combining first order killed mutants to create higher degree mutants is not a good way to evaluate and improve the quality of given set of test cases because the generated HOMs are easy to kill.

The HOMT1 and HOMT3 strategies seem to be better and offer more opportunities to improve the quality of given set of test cases, as MSI (and the number of killed mutants) decreased in comparison to HOMT2.

The experimental results indicated that, we should not use first order live mutants to create difficult (but possible) to kill higher order mutants. And using not-easy-to-kill mutants to generate higher order mutants is a promising method, which could be applied to the area of higher order mutation testing to evaluate and improve the quality of given set of TCs.

5 Threats to validity

Equivalent mutants constitute a threat to validity because the ratio of equivalent mutants in each strategy is unknown, while the problem of detecting equivalence between two mutants is an undecidable problem [7]. Furthermore, using five selected projects under test (PUTs) may not be representative of all Java programs in general and therefore, the results of the study may not be generalizable to all Java programs. Additionally the number of evaluated strategies is limited and, we think that further investigations would allow proposing new strategies for generating difficult (but possible) to kill higher order mutants. Applying other multi-objective optimization algorithms as well as the large PUTs is also needed to improve the obtained results.

Quang Vu Nguyen and Lech Madeyski, Intelligent Information and Database Systems: 8th Asian Conference, ACIIDS 2016, Da Nang, Vietnam, March 14-16, 2016, Proceedings, Part I, vol. 9621 of Lecture Notes in Artificial Intelligence, ch. Higher Order Mutation Testing to Drive Development of New Test Cases: An Empirical Comparison of Three Strategies, pp. 235–244. Springer, 2016. DOI: 10.1007/978-3-662-49381-6_23 (URL: http://dx.doi.org/10.1007/978-3-662-49381-6_23, Draft: http://madeyski.e-informatyka.pl/download/NguyenMadeyski16.pdf).

6 Conclusions and future work

We applied our objectives and fitness functions to multi-objective optimization algorithms for constructing HOMs from the set of generated FOMs in three ways. In the first one, we used all of the FOMs, in the second one, we used the FOMs, which were killed by at least one TC, while in the third one, we used not-easy-to-kill FOMs. The results indicated that applying multi-objective optimization in the area of higher order mutation testing to generate HOMs could be an interesting complementary approach to FOMT, but the strategy of selecting FOMs to build HOMs is of great importance. The strategy one should absolutely avoid is to build HOMs on a basis of killed FOMs (i.e., HOMT2). The alternative strategies HOMT1 and HOMT3, where HOMs are built on a basis of all FOMs give better results. The obtained results suggest the direction of further investigation, which could be a strategy where HOMs are build, for example, on a basis of live FOMs and/or HOMs of lower degree.

Applying multi-objective optimization algorithms to generate higher order mutants is a promising way for overcoming the limitations of mutation testing [5, 22]. In our previous work [22], the results of our experiment indicated that our approach is able to reduce the generated HOMs compared with FOMs as well as is useful in constructing higher order mutants. In this paper, we shed additional light on usefulness of higher order mutation strategies to drive development of new, high quality test cases.

In further research, we will investigate how process metrics [24, 25], based on development of test cases, combined with product metrics [26], based on mutation testing, can improve software defect prediction models we build in collaboration with our industrial partners [27, 28].

References

1. DeMillo, R.A., Lipton R.J., Sayward, F.G.: Hints on test data selection: help for the practicing programmer. IEEE Computer 11 (4), 34–41 (1978)
2. Hamlet, R.G.: Testing programs with the aid of a compiler. IEEE Transactions on Software Engineering SE-3 (4), 279–290 (1977)
3. Jia, Y., Harman, M.: Higher order mutation testing. Information and Software Technology 51, 1379–1393 (2009)
4. Harman, M., Jia, Y., Langdon, W. B.: A Manifesto for Higher Order Mutation Testing. In: Third International Conf. on Software Testing, Verification, and Validation Workshops, (2010)
5. Langdon, W.B., Harman, M., Jia, Y.: Efficient multi-objective higher order mutation testing with genetic programming. Journal of Systems and Software 83 (2010)

Quang Vu Nguyen and Lech Madeyski, Intelligent Information and Database Systems: 8th Asian Conference, ACIIDS 2016, Da Nang, Vietnam, March 14-16, 2016, Proceedings, Part I, vol. 9621 of Lecture Notes in Artificial Intelligence, ch. Higher Order Mutation Testing to Drive Development of New Test Cases: An Empirical Comparison of Three Strategies, pp. 235–244. Springer, 2016. DOI: 10.1007/978-3-662-49381-6_23 (URL: http://dx.doi.org/10.1007/978-3-662-49381-6_23, Draft: http://madeyski.e-informatyka.pl/download/NguyenMadeyski16.pdf).

6. Jia, Y., and Harman, M.: Constructing Subtle Faults Using Higher Order Mutation Testing. In: Proc. Eighth Int'l Working Conf. Source Code Analysis and Manipulation (2008)
7. Madeyski, L., Orzeszyna, W., Torkar, R., Józala, M.: Overcoming the Equivalent Mutant Problem: A Systematic Literature Review and a Comparative Experiment of Second Order Mutation. *IEEE Transactions on Software Engineering*, 40 (1), pp. 23-42, 2014. DOI: 10.1109/TSE.2013.44
8. Mresa, E.S., Bottaci, L.: Efficiency of mutation operators and selective mutation strategies: An empirical study. Software Testing, Verification and Reliability 9 (4) 205-232 (1999)
9. Papadakis, M., Malevris, N.: 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, pp. 90–99 (2010)
10. Vincenzi, A.M.R., Nakagawa, E.Y., Maldonado, J.C., Delamaro, M.E., Romero, R.A.F: Bayesian-learning based guidelines to determine equivalent mutants. International Journal of Software Engineering and Knowledge Engineering, 12 (6), 675–690 (2002)
11. Polo, M., Piattini, M., Garcia-Rodriguez, I.: Decreasing the Cost of Mutation Testing with Second-Order Mutants. Software Testing, Verification, and Reliability 19 (2), 111-131 (2008)
12. Nguyen, Q.V., Madeyski, L.: Problems of Mutation Testing and Higher Order Mutation Testing. In Do, T. and Le Thi, H. A. and Nguyen, N. T. (eds.) OCCSAMA 2014, Advanced Computational Methods for Knowledge Engineering, Advances in Intelligent Systems and Computing, vol. 282, pp. 157-172, Springer (2014). DOI: 10.1007/978-3-319-06569-4_12
13. Zhu, H., Hall, P.A.V., May, J.H.R.: Software Unit Test Coverage and Adequacy. ACM Computing Surveys 29 (4), 366-427 (1997)
14. Madeyski, L.: On the effects of pair programming on thoroughness and fault-finding effectiveness of unit tests. In: Muench, J. and Abrahamsson, P. (eds.) PROFES 2007, LNCS (Lecture Notes in Computer Science), vol. 4589, pp. 207-221. Springer, Heidelberg (2007). DOI: 10.1007/978-3-540-73460-4_20
15. Madeyski, L., The impact of pair programming on thoroughness and fault detection effectiveness of unit tests suites. Software Process: Improvement and Practice, 13 (3), 281-295 (2008). DOI: 10.1002/spip.382
16. Madeyski, L., The impact of test-first programming on branch coverage and mutation score indicator of unit tests: An experiment. Information and Software Technology, 52 (2), 169-184 (2010). DOI: 10.1016/j.infsof.2009.08.007
17. Madeyski, L., Radyk, N.: Judy - a mutation testing tool for Java. IET Software 4 (1), 32-42 (2010). DOI: 10.1049/iet-sen.2008.0038

Quang Vu Nguyen and Lech Madeyski, Intelligent Information and Database Systems: 8th Asian Conference, ACIIDS 2016, Da Nang, Vietnam, March 14-16, 2016, Proceedings, Part I, vol. 9621 of Lecture Notes in Artificial Intelligence, ch. Higher Order Mutation Testing to Drive Development of New Test Cases: An Empirical Comparison of Three Strategies, pp. 235–244. Springer, 2016. DOI: 10.1007/978-3-662-49381-6_23 (URL: http://dx.doi.org/10.1007/978-3-662-49381-6_23, Draft: http://madeyski.e-informatyka.pl/download/NguyenMadeyski16.pdf).

18. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A Fast and Elitist Multi objective Genetic Algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 6 (2), 182-197 (2002).
19. Deb, K., Jain, H., An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints, IEEE Transactions on Evolutionary Computation, 18 (4), 577-601 (2014).
20. Kollat, J.B., Reed, P.M., The Value of Online Adaptive Search: A Performance Comparison of NSGAII, ε-NSGAII and ε-MOEA, Carlos A. Coello Coello, Arturo Hernández Aguirre, Eckart Zitzler (Eds.), Evolutionary Multi-Criterion Optimization, Third International Conference, EMO 2005 Guanajuato, Mexico, March 9-11, 2005.
21. Deb, K., Mohan, M., Mishra, S., A Fast Multi-objective Evolutionary Algorithm for Finding Well-Spread Pareto-Optimal Solutions. KenGAL, Report No. 2003002. Indian Institute of Technology, Kanpur, India, 2003.
22. Nguyen, Q.V., Madeyski, L.: Searching for Strongly Subsuming Higher Order Mutants by Applying Multi-objective Optimization Algorithm. In: Le Thi, H.A., Nguyen, N.T., Do, T.V. (eds.) Advanced Computational Methods for Knowledge Engineering, Advances in Intelligent Systems and Computing, vol. 358, pp. 391-402. Springer (2015). DOI: 10.1007/978-3-319-17996-4_35
23. Purushothaman, R., Perry, D.E.: Toward Understanding the Rhetoric of small source code changes. IEEE Transactions on Software Engineering 31(6), 511-526 (2005).
24. Madeyski, L., Jureczko, M.: Which process metrics can significantly improve defect prediction models? An empirical study. Software Quality Journal 23(3), 393-422 (2015). DOI: 10.1007/s11219-014-9241-7
25. Jureczko, M., Madeyski, L.: A review of process metrics in defect prediction studies. Metody Informatyki Stosowanej 30(5), 133-145 (2011). http://madeyski.e-informatyka.pl/download/Madeyski11.pdf
26. Jureczko, M., Madeyski, L.: Towards identifying software project clusters with regard to defect prediction. In: Proceedings of the 6th International Conference on Predictive Models in Software Engineering (PROMISE '10). ACM, New York, NY, USA, , Article 9, 9:1-9:10. DOI: 10.1145/1868328.1868342
27. Madeyski, L., Majchrzak, M.: Software Measurement and Defect Prediction with DePress Extensible Framework. Foundations of Computing and Decision Sciences 39 (4), 249-270 (2014). DOI: 10.2478/fcds-2014-0014
28. Hryszko J., Madeyski, L.: Bottlenecks in software defect prediction implementation in industrial projects. Foundations of Computing and Decision Sciences 40 (1), 17-33 (2015). DOI: 10.1515/fcds-2015-0002
29. Nguyen, Q.V., Madeyski, L.: Empirical evaluation of multi-objective optimization algorithms searching for higher order mutants. Cybernetics and Systems: An

Quang Vu Nguyen and Lech Madeyski, Intelligent Information and Database Systems: 8th Asian Conference, ACIIDS 2016, Da Nang, Vietnam, March 14-16, 2016, Proceedings, Part I, vol. 9621 of Lecture Notes in Artificial Intelligence, ch. Higher Order Mutation Testing to Drive Development of New Test Cases: An Empirical Comparison of Three Strategies, pp. 235–244. Springer, 2016. DOI: 10.1007/978-3-662-49381-6_23 (URL: http://dx.doi.org/10.1007/978-3-662-49381-6_23, Draft: http://madeyski.e-informatyka.pl/download/NguyenMadeyski16.pdf).

International Journal, (2016) (accepted). DOI: 10.1080/01969722.2016.1128763
URL: http://madeyski.e-informatyka.pl/download/NguyenMadeyski16CS.pdf

Quang Vu Nguyen and Lech Madeyski, Intelligent Information and Database Systems: 8th Asian Conference, ACIIDS 2016, Da Nang, Vietnam, March 14-16, 2016, Proceedings, Part I, vol. 9621 of Lecture Notes in Artificial Intelligence, ch. Higher Order Mutation Testing to Drive Development of New Test Cases: An Empirical Comparison of Three Strategies, pp. 235–244. Springer, 2016. DOI: 10.1007/978-3-662-49381-6_23 (URL: http://dx.doi.org/10.1007/978-3-662-49381-6_23, Draft: http://madeyski.e-informatyka.pl/download/NguyenMadeyski16.pdf).