Using Evolutionary Computation to Improve Mutation Testing

Pedro Delgado-Pérez[1], Inmaculada Medina-Bulo[1], and Mercedes G. Merayo[2]

[1] Departamento de Ingeniería Informática, Escuela Superior de Ingeniería, Universidad de Cádiz, Cádiz, España,
{pedro.delgado,inmaculada.medina}@uca.es
[2] Departamento de Sistemas Informáticos y Computación Universidad Complutense de Madrid, Madrid, Spain,
mgmerayo@fdi.ucm.es

Abstract The work on mutation testing has attracted a lot of attention during the last decades. Mutation testing is a powerful mechanism to improve the quality of test suites based on the injection of syntactic changes into the code of the original program. Several studies have focused on reducing the high computational cost of applying this technique and increasing its efficiency. Only some of them have tried to do it through the application of genetic algorithms. Genetic algorithms can guide through the generation of a reduced subset of mutants without significant loss of information. In this paper, we analyse recent advances in mutation testing that contribute to reduce the cost associated to this technique and propose to apply them for addressing current drawbacks in Evolutionary Mutation Testing (EMT), a genetic algorithm based technique with promising experimental results so far.

Keywords: Software testing, mutation testing, genetic algorithms.

1 Introduction

Mutation testing [20] is an effective technique to help improve the fault detection capability of a test suite. In order to estimate this capability, we measure the extent to which the developed test suite is able to detect different faults injected into the program. These faults are introduced by means of *mutation operators* that generate *mutants*. Detecting a mutation means that the test suite can distinguish the behaviour of the mutant and the original program. In this case, the mutant is said to be *killed*. Otherwise, when the outputs of the original and the mutant are equal, we say that the mutant is *alive*. However, some alive mutants are *equivalent* to the original program and cannot be killed by any test case. Mutation testing is a powerful testing technique but it has a high computational cost due to the potentially large number of mutants that can be generated, in particular, when the program requires a high compilation and test execution time. Therefore, it is required to propose approaches that, on the one hand, reduce the size of the set of mutants and, on the other hand, provide a high level

of fault detection. As a result, several methods have been proposed to alleviate as much as possible this problem [19], including search-based techniques.

In this paper we focus on the application of the approaches proposed in mutation testing to Evolutionary Mutation Testing, a technique for the selection of a reduced set of mutants based on a genetic algorithm [8]. It will allow us to improve the promising results reported by this methodology to date for WS-BPEL compositions [6] and C++ object-oriented systems [3]. We propose to integrate the following approaches:

- **Selective mutation based on the quality metric** [7], which aims at identifying mutants that could be excluded without losing effectiveness of the test suite. We can choose either to discard a subset of mutation operators (operator-based selective mutation) or give preference to the selection of mutants from the most valuable operators (rank-based mutant selection).
- **A multi-objective approach** that drives the search towards finding, not only undetected mutants, but also mutants with a great coverage impact [14] and well spread through the code [18].
- **Trivial Compiler Equivalence** [15], a mechanism to detect equivalent mutants when they have identical machine code as the original program.

The rest of the paper is organized as follows: Section 2 surveys the use of genetic algorithms in mutation testing, and more specifically in EMT. Then, we explain the proposals based on each of these findings and comment the benefits of their application in Sections 3, 4 and 5. Finally, we present the conclusions derived from this paper.

2 Genetic Algorithms in Mutation Testing

Genetic algorithms have been applied to software testing since many years ago [21]. With regard to mutation testing, genetic algorithms have been successfully investigated and incorporated into different systems for the generation of test cases [16]. However, they have also shown to be useful in mutant generation to increase the efficiency of mutation testing. We can remark the following advances:

- The evolution in parallel of the population of mutants and test cases using different genetic operators [1,12].
- Several studies have shown that genetic algorithms can provide better results than other search-based techniques [10,13] in finding interesting higher order mutants.
- Genetic algorithms have been used to select a subset of mutants [18] and also a subset of mutation operators [2] with the goal of minimising the loss of information.

Evolutionary Mutation Testing [6] was proposed to select a subset of mutants with the help of an evolutionary algorithm. Specifically, a genetic algorithm was implemented in the *GAmera* tool [5] to search for *strong mutants* that can help derive new test cases. Strong mutants fall in one of these categories:

– *Potentially equivalent mutants*: mutants that are not detected by the test suite.
– *Difficult to kill mutants*: mutants that are only killed by a specific test case that does not kill any other mutants.

The fitness function gives the highest value to those mutants killed by few test cases and those test cases killing few other mutants at the same time. In this way, strong mutants are the best valued by the fitness function. The mutants in the new generations are then:

– Derived from those mutants with a high fitness thanks to reproductive operators (mutation and crossover). This should lead to the generation of strong mutants in new generations.
– Produced by a random algorithm.

The studies originally conducted on EMT [6] reported that this technique is able to find all strong mutants generating 15% less mutants than random selection for WS-BPEL compositions. Recently, the experiments performed with C++ systems and class-level mutation operators [3] supported that EMT is, in fact, more effective than random mutant selection, though the difference was not as significant as in the experiments with WS-BPEL. This fact might be motivated by the way that strong mutants are distributed over the search space, and reveals that further research is required to enhance the effectiveness of EMT.

3 Quality metric for selective mutation

3.1 Background

Some mutants have greater potential than others to improve the quality of the test suite. They can guide the tester through the definition of *high-quality test cases* [4], that is, test cases that are able to find non-trivial faults. Regarding mutation operators, those ones that generate mutants that can be killed, mostly, by a default "happy path" test case, are not useful. These considerations are embodied in the quality metric devised in [7]. The formula of the quality metric of a mutant Q_m is defined as follows:

$$Q_m = \begin{cases} 0, & m \in E \\ 1 - \dfrac{1}{(|M| - |E|) \cdot |T|} \displaystyle\sum_{t \in K_m} |C_t|, & m \in D \end{cases} \tag{1}$$

Where:

– M is the set of valid mutants.
– E is the set of equivalent mutants.
– D is the set of killed mutants.
– T is an adequate and minimal test suite, i.e., a test suite of the minimal size that kills all non-equivalent mutants.

– K_m is the set of test cases that kill the mutant m.
– C_t is the set of mutants killed by the test case t.

This quality metric punishes the existence of equivalent mutants as well as takes into account a twofold criteria regarding killed mutants:

– The number of test cases that kill a mutant: the fewer test cases kill the mutant, the better.
– The number of mutants that a particular test case kill: the fewer the mutants killed by a test case, the better. This property is valuable because we only have a reduced subset of mutants that can induce the generation of that test case.

Next, we find the definition of the quality of a mutation operator Q_o as the mean of the quality of the set of mutants generated by the operator o (M_o):

$$Q_o = \frac{1}{|M_o|} \sum_{m \in M_o} Q_m \tag{2}$$

Recently, this metric has been used to determine the capacity of mutation operators to help the tester enhance the fault detection power of the test suite with high-quality test cases. The mutation operators were sorted in a ranking according to their quality. Then, two different selective strategies were applied taking into account the ranking:

– **Operator-based selective mutation**: this strategy selected mutants generated by a subset of mutation operators.
– **Rank-based mutant selection**: this strategy favoured the selection of the mutants generated by the top ranked operators. The results of the rank-based strategy showed that this method for the selection of mutants offers a better outcome than the random selection of mutants.

Both strategies were evaluated by measuring the percentage of test cases that would not be generated due to the dismissed mutants. This approach was successful in reducing the number of mutants without a meaningful loss of effectiveness.

3.2 Improvements applying selective mutation based on quality metrics

Selective mutation following operator rankings improves the test suite through the selection of a subset of all the mutants. Initially, we use it to analyse and determine which are the best and the worst-valued operators in order to incorporate the gained knowledge into the mutation tool. Then, EMT can take advantage of this information to reduce the set of operators to be applied and, therefore, the cost. As a result, the combination of both techniques can be used to further reduce the cost. These results can be used in two different ways:

- *First proposal*: Instead of generating mutants from all mutation operators, EMT only generates mutants from the best-valued operators following the ranking based on the quality metric. This proposal requires to determine if a mutation operator should be discarded (based on previous studies with the set of mutation operators applied).
- *Second proposal:* Instead of generating mutants with the same probability, the mutants are selected in a rank-based manner: the probability of selecting mutants from the best-valued operators is higher. This criterium is applied to the subset of mutants randomly produced in each generation of the genetic algorithm.

Mutant-based selective mutation and operator-based selective mutation have shown to be useful to reduce the cost of mutation testing while retaining effectiveness. However, in the experiments presented in [4], mutant-based selection yielded better results than operator-based selection when applied to mutation operators at the class-level in C++. However, the analysis of traditional operators has usually revealed great redundancy among operators and, consequently, a subset of operators could subsume the rest. As a consequence, it is unclear which strategy shows a better performance, so the proposal should be selected depending on the nature of the set of mutation operators based on previous empirical results.

3.3 Benefits

Both proposals modify the performance of EMT in the following ways:

- *First proposal*: By removing the operators at the bottom of the classification, the efficiency of EMT is improved because it will avoid that the genetic algorithm produces low-quality mutants. Since the quality metric punishes equivalent mutants, which decreases the value of a mutation operator, we will be also preventing the selection of equivalent mutants. In general, the genetic algorithm finds quicker those mutants that can lead to the enhancement of the test suite. These mutants can be labelled as *resistant mutants*.
  The reason that non-equivalent mutants remain alive is either (a) the test suite does not cover the mutant or (b) the test suite covers the mutant but it is not able to reveal its mutation. Since the quality metric is devoted to the generation of high-quality test cases, the refinement is achieved with a large proportion of mutants that fall in the case (b), which are the most interesting because they are not easy to design.
- *Second proposal:* The first proposal involves the risk of eliminating the possibility that some mutants of a high quality derived from low-quality operators are generated. The application of the rank-based strategy removes that risk, especially in those cases when it is not easy to find a sufficient set of mutation operators. The same benefits mentioned for the first proposal hold, but there is a higher probability that some low-quality mutants are generated.

4 Multi-objective Evolutionary Mutation Testing

4.1 Background

One of the major drawbacks of mutation testing is the presence of equivalent mutants. They cannot be completely discarded in general because this is an undecidable problem. Therefore, they have to be detected when the alive mutants are analysed, which is a time-consuming task. Several researchers have proposed different techniques to identify and remove equivalent mutants [11,9]. Recently, several works have studied the impact that mutations have on the code coverage in order to mitigate the effects of the equivalence [14,17]. Intuitively, those mutations which cause a great impact on the coverage of the test suite execution are less likely to produce an equivalent mutant. In addition, there should not be parts of the code without mutations, because it would avoid that those fragments are analysed with mutation testing [18].

The results obtained when applying an operator-based and a mutant-based selective strategy have been compared [23]. Namely, two different mutant-based strategies are applied: *One-round* random selection (mutants selected with the same probability) and *Two-round* random selection (the operator that generates the mutant is selected with the same probability). Given that Two-round random selection yielded better results than One-round random selection, it is plausible to think that each mutation operator is useful to address a different feature. Similar results were reported in other experiments using class-level mutation operators [4]. The experiments reported in [22] also suggest that sampling mutants from each method of the program and from each mutation operator performs better than One-round random selection.

4.2 New Objectives

Based on the aforementioned studies and their results, EMT should follow a multi-objective approach. We have detected three aspects that should be considered in the fitness function of the genetic algorithm:

1. **Maximise the coverage impact**. EMT currently assigns the highest fitness to potentially equivalent mutants, but it cannot distinguish between equivalent mutants and resistant mutants. Analysing the coverage impact of the mutants will help during the selection of non-equivalent mutants with the highest probability.
2. **Maximise the scattering in the code**. The genetic algorithm currently selects mutations without taking care of the location in which it is injected. As a consequence, the reduced subset of mutants might not be dealing with some parts of the code appropriately. Therefore, it is preferable spreading mutations all over the code so that all code items are covered [18,22].
3. **Maximise the scattering in the set of operators**. We foster the generation of mutants from all mutation operators, favouring the selection of mutants from operators barely applied so far. This aspect is not currently taken into account by the genetic algorithm. This objective aligns with recent findings in the mutation testing literature [4,22,23].

4.3 Benefits

Next, we present the benefits of incorporating each of these aspects into EMT:

1. *Coverage impact*: Penalisation of those mutants with the highest probability of resulting in equivalent mutants prevents from selecting equivalent mutants for reproduction and new equivalent mutants being generated.
2. *Scattering in the code*: A program can be divided into different classes, which count with different methods comprised of multiple statements. Furthermore, in an object-oriented program, some methods or even blocks are directly associated to specific object-oriented features, such as constructors or exceptions. Therefore, a fitness function aware of the coverage will avoid that the subset of selected mutants concentrates in the same area of the code.
3. *Scattering in the set of operators*: Each of the operators affects to different features, especially in the case of class-level operators. Consequently, strong mutants might be generated by a large subset of mutation operators. As a result of this improvement, the gap between the percentage of mutants generated by each of the operators is less significant.

It is worth noting that the weight of the application of each of aspects can be parametrised in order to prioritise the most convenient for the specific context.

5 Trivial Compiler Equivalence

5.1 Background

As aforementioned, the existence of equivalent mutants is one of the main problems in mutation testing. Recently, a new technique, called Trivial Compiler Equivalence (TCE), has been proposed to detect some equivalent mutants automatically [15]. This technique has shown the ability to reduce an average of 30% the set of equivalent mutants in programs coded in C.

TCE combines the use of the compiler *gcc* and the utility *diff*: *gcc* is used to compile the code and generate an optimised executable, and *diff* allows to compare these executables to search for equivalences between different versions of the program. This mechanism is able to detect two types of mutants:

- **A subset of equivalent mutants**: a mutant is identified as equivalent when there is no difference between the binary file of the original program and a mutant.
- **A subset of duplicated mutants**: a mutant is identified as duplicated when the binary file derived from the mutant is equal to the the binary file derived from another mutant.

The experimental results showed that the application of this technique is affordable in terms of execution time. However, the most expensive task in TCE is the compilation time, which increases with the level of optimisation selected. The detection of equivalent and duplicated mutants is not significant when compared to the compilation time.

5.2 Improvements applying TCE

The genetic algorithm in EMT searches for potentially equivalent mutants, which have not been detected by the current test suite. Hopefully, this type of mutants will help derive new test cases. However, potentially equivalent mutants can also turn out to be equivalent, being unable to distinguish them from those that lead us to improve the test suite. Therefore, some of those equivalent mutants can be automatically detected when EMT is combined with TCE. Given that TCE is based on the compiler *gcc*, the technique is available for those mutation tools that apply to programming languages within the collection of compilers in *gcc*, like *GiGAn* for C++ [3].

We propose two different options for applying EMT in conjunction with TCE:

- *First proposal*: Applying TCE before the execution of EMT. In this case, all mutants are compiled to create an executable and TCE compares them with the executable of the original program. Those mutants detected by TCE as equivalent are then marked to avoid that they can be used during the execution of EMT.
- *Second proposal*: Applying TCE during the execution of EMT. In this case, the mutants selected in each generation are analysed using TCE. Those mutants identified as equivalent are penalised by means of the fitness value assigned to them.

The detection of duplicated mutants could also be beneficial to avoid the selection of mutants that can lead to the generation of the same test case. However, for the manual generation of test cases, it might be useful sometimes to count with redundant mutants: a tester may find difficult to kill a mutant but may find more feasible to produce a test case for another mutant which kills both mutants.

5.3 Benefits

The benefits of implementing these strategies are:

- *First proposal*: Applying TCE to all mutants from the beginning, we avoid that EMT selects equivalent mutants that can be detected by TCE. However, we are wasting time trying to detect equivalent mutants which might never be selected by EMT, so this proposal is not recommended for programs with a high compilation time. On the contrary, it is very convenient when there is evidence that TCE is able to detect a high percentage of equivalent mutants for the set of mutation operators applied.
- *Second proposal*: In this case, we only apply TCE when finding a potentially equivalent mutant during the execution of EMT. Although this proposal decreases the computational cost of applying TCE, it does not avoid selecting equivalent mutants occasionally.

Nevertheless, we have to take into account the restrictions imposed by TCE:

– TCE has only been applied to C programs. Thus, it is not clear that, if it is applied to other languages or other type of mutation operators, we will obtain the same detection power.
– Under the premise that an equivalent mutant and a resistant mutant are slightly different, it is possible that some equivalent mutants can guide the genetic algorithm on the selection of resistant mutants in new generations. Therefore, removing or penalising equivalent mutants can impact the search for potentially equivalent mutants.

6 Conclusions

In this paper, we have analysed the current operation mode of the technique called Evolutionary Mutation Testing. Despite evidence of its usefulness, based on the empirical results obtained from its application to programs in different languages, from a review of the mutation literature it becomes clear that further improvements can be made. In particular, EMT does not solve the hardest task of mutation testing: the detection of equivalent mutants. With the goal of overcoming this drawback, we propose to consider the use of additional information in the calculation of the fitness function and the application of TCE, a technique for the automated detection of some equivalent mutants. The new fitness function also promotes the generation of mutants from all mutation operators and covering all the code.

We also propose the generation of mutants following an operator-based and/or a rank-based selective approach, which also penalises the generation of mutants from operators usually producing many equivalent mutants. However, the application of these two selective strategies should be explored further with different sets of mutation operators to know the extent to which mutant-based selection is superior to operator-based selection, as previous studies suggest. Rank-based selective mutation is based on the rank selection method used in genetic algorithms; it would also be interesting to compare its results with different mutant-based selection techniques, such as roulette wheel selection, tournament selection or stochastic universal sampling.

In spite of proposing several improvements for EMT, there is still room for other ones. For instance, we should try to find a way to avoid that all test cases are executed to calculate the fitness function, especially in those cases when the test execution time is high.

Acknowledgements: Paper partially funded by the research scholarship PU-EPIF-FPI-PPI-BC 2012-037 (University of Cádiz) and by Spanish government projects DArDOS (TIN2015-65845-C3-3-R (MINECO/FEDER)), SICOMORo-CM (S2013/ICE-3006) and the Excellence Network SEBASENet (TIN2015-71841-REDT (MINECO)).

References

1. Adamopoulos, K., Harman, M., Hierons, R.M.: How to overcome the equivalent mutant problem and achieve tailored selective mutation using co-evolution. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004). pp. 1338–1349 (2004), http://dx.doi.org/10.1007/978-3-540-24855-2_155
2. Banzi, A.S., Nobre, T., Pinheiro, G.B., Árias, J.C.G., Pozo, A., Vergilio, S.R.: Selecting mutation operators with a multiobjective approach. Expert Systems with Applications 39(15), 12131–12142 (2012), http://dx.doi.org/10.1016/j.eswa.2012.04.041
3. Delgado-Pérez, P., Medina-Bulo, I., Segura, S., Domínguez-Jiménez, J.J., García-Domínguez, A.: GiGAn: Evolutionary mutation testing for C++ object-oriented systems. In: The 32nd ACM Symposium On Applied Computing (SAC 2017).
4. Delgado-Pérez, P., Segura, S., Medina-Bulo, I.: Assessment of C++ object-oriented mutation operators: A selective mutation approach. Software Testing, Verification and Reliability (2017), http://dx.doi.org/10.1002/stvr.1630
5. Domínguez-Jiménez, J.J., Estero-Botaro, A., García-Domínguez, A., Medina-Bulo, I.: GAmera: an automatic mutant generation system for WS-BPEL compositions. In: Proceedings of the 7th IEEE European Conference on Web Services. pp. 97–106. IEEE Computer Society Press, Eindhoven, The Netherlands (Nov 2009), http://dx.doi.org/10.1109/ECOWS.2009.18
6. Domínguez-Jiménez, J.J., Estero-Botaro, A., García-Domínguez, A., Medina-Bulo, I.: Evolutionary mutation testing. Information and Software Technology 53(10), 1108–1123 (Oct 2011), http://dx.doi.org/10.1016/j.infsof.2011.03.008
7. Estero-Botaro, A., Palomo-Lozano, F., Medina-Bulo, I., Domínguez-Jiménez, J.J., García-Domínguez, A.: Quality metrics for mutation testing with applications to WS-BPEL compositions. Software Testing, Verification and Reliability 25(5-7), 536–571 (2015), http://dx.doi.org/10.1002/stvr.1528
8. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edn. (1989)
9. Hierons, R., Harman, M., Danicic, S.: Using program slicing to assist in the detection of equivalent mutants. Software Testing, Verification and Reliability 9(4), 233–262 (1999), http://dx.doi.org/10.1002/(SICI)1099-1689(199912)9:4<233::AID-STVR191>3.0.CO;2-3
10. Jia, Y., Harman, M.: Constructing subtle faults using higher order mutation testing. In: Proceedings of the Eighth IEEE International Working Conference on Source Code Analysis and Manipulation, 2008. pp. 249–258 (Sept 2008), http://dx.doi.org/10.1109/SCAM.2008.36
11. Offutt, A.J., Pan, J.: Detecting equivalent mutants and the feasible path problem. In: Proceedings of the Eleventh Annual Conference on Computer Assurance, Systems Integrity. Software Safety. Process Security (COMPASS 1996). pp. 224 –236 (June 1996), http://dx.doi.org/10.1109/CMPASS.1996.507890
12. de Oliveira, A.A.L., Camilo-Junior, C.G., Vincenzi, A.M.R.: A coevolutionary algorithm to automatic test case selection and mutant in mutation testing. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2013). pp. 829–836 (June 2013), http://dx.doi.org/10.1109/CEC.2013.6557654
13. Omar, E., Ghosh, S., Whitley, D.: Homaj: A tool for higher order mutation testing in AspectJ and Java. In: Proceedings of the IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops (ICSTW 2014). pp. 165–170 (March 2014), http://dx.doi.org/10.1109/ICSTW.2014.19

14. Papadakis, M., Delamaro, M., Traon, Y.L.: Mitigating the effects of equivalent mutants with mutant classification strategies. Science of Computer Programming 95, Part 3, 298–319 (2014), http://dx.doi.org/10.1016/j.scico.2014.05.012, special Section: ACM SAC-SVT 2013 + Bytecode 2013
15. Papadakis, M., Jia, Y., Harman, M., Le Traon, Y.: Trivial compiler equivalence: A large scale empirical study of a simple, fast and effective equivalent mutant detection technique. In: Proceedings of the 37th International Conference on Software Engineering - Volume 1 (ICSE 2015). pp. 936–946. IEEE Press, Piscataway, NJ, USA (2015), http://dx.doi.org/10.1109/ICSE.2015.103
16. Pargas, R.P., Harrold, M.J., Peck, R.R.: Test-data generation using genetic algorithms. Software Testing, Verification and Reliability 9(4), 263–282 (1999), http://dx.doi.org/10.1002/(SICI)1099-1689(199912)9:4<263::AID-STVR190>3.0.CO;2-Y
17. Schuler, D., Zeller, A.: Covering and uncovering equivalent mutants. Software Testing, Verification and Reliability 23(5), 353–374 (2013), http://dx.doi.org/10.1002/stvr.1473
18. Schwarz, B., Schuler, D., Zeller, A.: Breeding high-impact mutations. In: Proceedings of the 4th IEEE International Conference on Software Testing, Verification, and Validation Workshops (ICSTW 2011). pp. 382–387 (2011), http://dx.doi.org/10.1109/ICSTW.2011.56
19. Usaola, M., Mateo, P.: Mutation testing cost reduction techniques: A survey. IEEE Software 27(3), 80–86 (2010), http://dx.doi.org/10.1109/MS.2010.79
20. Woodward, M.R.: Mutation testing - its origin and evolution. Information and Software Technology 35(3), 163–169 (Mar 1993), http://dx.doi.org/10.1016/0950-5849(93)90053-6
21. Xanthakis, S., Ellis, C., Skourlas, C., Le Gall, A., Katsikas, S., Karapoulios, K.: Application of genetic algorithms to software testing. In: Proceedings of the 5th International Conference on Software Engineering and Applications. pp. 625–636 (1992)
22. Zhang, L., Gligoric, M., Marinov, D., Khurshid, S.: Operator-based and random mutant selection: Better together. In: Proceedings of the IEEE/ACM 28th International Conference on Automated Software Engineering (ASE 2013). pp. 92–102 (Nov 2013), http://dx.doi.org/10.1109/ASE.2013.6693070
23. Zhang, L., Hou, S.S., Hu, J.J., Xie, T., Mei, H.: Is operator-based mutant selection superior to random mutant selection? In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1 (ICSE 2010). pp. 435–444, ACM, New York, NY, USA (2010), http://dx.doi.org/10.1145/1806799.1806863