Proceedings of the Federated Conference on Computer Science and Information Systems pp. 1593–1596
DOI: 10.15439/2016F82
ACSIS, Vol. 8. ISSN 2300-5963

Applying Mutation Testing for Assessing Test Suites Quality at Model Level

Joanna Strug
Faculty of Electrical and Computer Engineering, Cracow University of Technology

ul. Warszawska 24, 31-155 Krakow, Poland
Email: pestrug@cyf-kr.edu.pl

***Abstract*—Models are commonly used in software testing to select test suites. Application of mutation testing at a model level can contribute to reliable and early assessment of the quality of the test suites. It can also support selection of test suites achieving high fault detection rates. The main issue related to using mutation testing at the early development stage is to determine how reliably the quality of test suites can be measured at the model level. The research presented in this paper addresses this problem for object-oriented systems. It focuses on describing an experiment aiming at comparing results of applying mutation testing at a model level with results of applying this technique at an implementation level and presents and discusses the outcomes of the experiment. The paper presents also mutation operators applicable at the model level.**

I. Introduction

MODELS play an important role in developing object-oriented software systems. They are also commonly used by researchers and practitioners involved in software testing as a source of data for selecting tests [25]. As the main goal of software testing is to detect faults in a tested system, an adequate assessment of suites of tests provided by any test generation approach is essential. Mutation testing is a well established techniques that helps to assess the quality of test suites with regard to their ability to detect faults [7]. To assess test suite quality, by means of this technique, a number of faulty versions of the system (called mutants) is generated, by introducing small changes into the original system, and executed against tests from the suite. The ratio of the number of mutants detected (killed) by these tests over the total number of non-equivalent mutants (called a mutation score for a test suite) determines the test suite ability to detect faults. The higher the mutation score for a test suite, the better the suite is in detecting faults. Although mutation testing is an effective test suite assessment technique, its application area is mostly limited to implementation level and high computational costs still prevent it from becoming a practical approach.

Application of mutation testing at a model level could be a good alternative or at least a valuable addition to the current practice in assessing the quality of test suites derived from models. It would allow to use the most reliable assessment technique at the same early level and may also lower the costs of generation and execution of mutants, as system models are less complex than the implementations.

However, two issues should be considered before applying the approach in practice:

1) choice of models for representing a system, and
2) evaluation of the reliability of results of a model level test suite quality assessment.

The research presented in this paper concerns object-oriented systems, therefore UML/OCL class diagrams were used to describe the systems at the model level.

The second issue is essential, when increasing the level of abstraction at which mutation testing is applied. A model represents only certain aspects of a final system, thus it is not possible to predict all implementation level faults based only on faults that appear at a model level. Moreover, faults introduced into a model target only features of the model, not of the language used to implement the system, and hence application of mutation testing at a model level assesses a test suite ability to detect faults specific to that level. Even in context of object-oriented systems modeled with UML/OCL and implemented in an object-oriented language, there are significant differences between both description formalisms. Thus, it is unclear if test suite assessment results (provided in the form of a mutation score) obtained at model level are sufficiently reliable to be accepted as a measure of a test suite quality in terms of its ability to detect faults in a final, implemented system. To the author best knowledge, the problem has not been studied before. The paper presents an approach to model level, UML/OCL-based assessment of test suite quality and describes an experiment carried out to address the problem. It provides a description of mutation operators applicable to UML/OCL models, the procedures for conducting the experiment and its results.

II. Related Work

Mutation testing was originally introduced at the implementation level [7] and the majority of papers describing different aspects of mutation testing dealt with problems concerning implementation level mutation only (a survey of such papers can be found in [11]). However, application of mutation testing at model level seems to gain popularity [1], [2], [16], [18], [19], [23], [24]. Authors of the approaches have focused mainly on selecting tests, but some of them have addressed also the problem of assessing tests at the model level [24], [23] and discussed selected aspects related to the problem of

978-83-60810-90-3/$25.00

assessing tests quality at different levels [23]. Although the problem considered by authors of the work in [23] is partially related with the one studied in this work, their approach is not applicable in the context of object-oriented systems, because the formalisms used in the paper cannot support adequately object-oriented aspects of the systems.

Relevant to this research are mainly papers describing application of mutation testing to UML and OCL based models [1], [2], [14], [16] and papers providing information that can help to design mutation operators applicable to UML and OCL. The set of UML/OCL related mutation operators introduced by the author in [17] was developed based on the fault taxonomy for UML [9], traditional mutation operators, operators adapted from other formalisms [8], [15] and operators defined for specifications [5], [12] and contracts [12] and was supplemented with five new OCL-specific operators.

The work presented in this paper differs from other works concerning model or implementation level mutation testing, as it targets UML/OCL class diagrams (standard models in developing object-oriented systems) and attempts to find out if test suites ability to detect implementation level faults can be assessed reliable at the model level.

III. Experimental Evaluation of a Reliability of Model Level Test Suite Quality Assessment Results

The goal of the research was to determine how reliably one can assess test suite quality, in terms of its ability to detect real faults in an object-oriented software system, by assessing the test suite using mutation testing at a the model level. Empirical studies on mutation testing have provided evidences that application of the technique at the implementation level provides adequate measurement of test suite quality (in terms of its ability to detect real faults in a final, implemented system) [6], [13]. Thus, the implementation level measurements can be referred to to determine the reliability of the test suite quality assessment results obtained at the model level.

A. Experimental measures

Application of mutation testing provides a mutation score for a test suite. The mutation score is a quantitative measure of the suite ability to detect mutants. For the rest of the paper let T denote a test suite, $MS_{IL}(T)$ denote a mutation score calculated for T at the implementation level and $MS_{ML}(T)$ denote a mutation score calculated for T at the model level.

Implementation level mutation score $MS_{IL}(T)$ for a test suite T expresses its ability to detect implementation level mutants (and thus their ability to detect real faults) and is defined in the following way:

$$MS_{IL}(T) = \frac{MI_D}{MI_T}, where$$

- MI_D is the number of implementation level mutants detected by T,
- MI_T is the total number of non-equivalent, implementation level mutants.

Model level mutation score $MS_{ML}(T)$ for a test suite T expresses its ability to detect model level mutants and is defined in the following way:

$$MS_{ML}(T) = \frac{MM_D}{MM_T}, where$$

- MM_D is the number of model level mutants detected by T,
- MM_T is the total number of non-equivalent, model level mutants.

The reliability of a model level test suite assessment result is measured for a test suite T by comparing the value of $MS_{ML}(T)$ with the value of $MS_{IL}(T)$.

B. Mutation Operators

Mutation operators are defined as transformation rules that produce faulty versions (so called mutants) of a program or a model [7]. Each operator can produce a number of mutants by changing instances of some construction of the formalism to which the operator is applied.

Within the work generation of mutants was controlled by two sets of mutation operators. At the implementation level mutation operators designed for Java and implemented in mujava [15] were used, and at the model level operators modifying UML/OCL class diagrams were used. The second set was divided into two groups: class diagram related mutation operators, and OCL related mutation operators.

The first group consists of the following operators:

- Hiding attribute deletion (IHD) - deletes in a subclass an attribute having the same name and type as an attribute in a parent class,
- Hiding attribute insertion (IHI) - inserts in a subclass an attribute having the same name and type as an attribute in a parent class,
- Attribute multiplicity change (CAMC) - changes a multiplicity of an attribute,
- Operation arguments order change (OAO) - changes the order of arguments in an operation definition,
- Operation arguments type replacement (ADR) - changes a declared type of a method argument to the parent of the originally declared type,
- Overriding operation deletion (IOD) - deletes an overriding operation in a subclass,
- Generalization association deletion (GAD) - deletes a generalization association between two classes,
- Generalization association direction change (GDC) - changes a direction of a generalization association,
- Association type replacement (ATR) - replaces a type of an association with another type,
- Association end multiplicity change (EMC) - changes multiplicity of an association end to other one,
- Association end class replacement (ECR) - replaces an association end class with a parent class or a subclass,
- Association role swap (ARS) - swaps role names of two associations between the same two classes or their subclasses.

The second group consists of the following operators:

- Operand Replacement Operator (ORO) - replaces an operand with another one, applies also for components of a navigation path,
- Arithmetic Operator Replacement (AOR) - replaces a binary arithmetic operator with another one,
- Arithmetic Operator Insertion (AOI) - inserts an unary arithmetic operator,
- Arithmetic Operator Deletion (AOD) - deletes an unary arithmetic operator,
- Relational Operator Replacement (ROR) - replaces a relational operator with another one,
- Conditional Operator Replacement (COR) - replaces a conditional operator with another one, supports operators: and, or, xor,
- Conditional (unary) Operator Insertion (COI) - inserts an unary conditional operator (not),
- Conditional (unary) Operator Deletion (COD) - deletes an unary conditional operator (not),
- @pre Deletion (POD) - deletes @pre operator,
- @pre Insertion (POI) - inserts @pre operator,
- Collection Operation Replacement (OCR) - replaces an invocation of a collection operation with another one,
- Collection Operation Deletion (OCD) - deletes an invocation of a collection operation,
- Contextual Instance Replacement (CIR) - replaces a contextual instance with another one.

C. Experimental procedures

The experiment was divided into two stages (Fig. 1):

1) model level test suites quality assessment, and
2) implementation level test suites quality assessment.

The experiment was performed on six experimental test suites provided for two object-oriented systems. The experiment, at each stage, was carried out following the same scenario, but dealt with the systems at different levels of abstraction (i.e. models or implementations) and was supported by different tools.

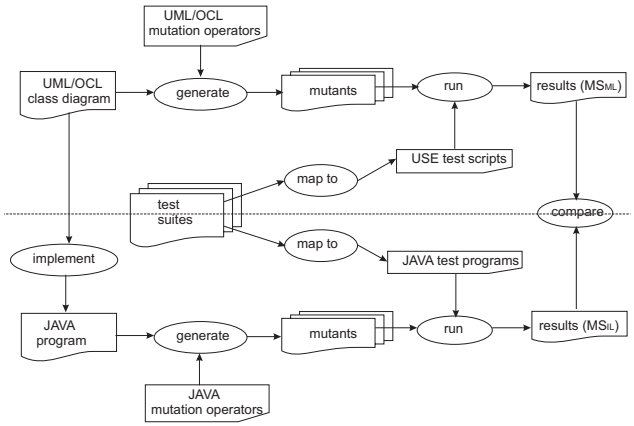

Fig. 1. An outline of the experiment

In the first stage each system was specified in a form of a UML/OCL class diagram, according to USE notation and each test suite was described as a USE command script [10]. Next, for each system, mutants of the system model were manually generated by applying mutation operators designed for UML/OCL models, and executed against tests from the test suites prepared for the system. The undetected mutants were then analyzed manually to remove the equivalent ones and for each test suite T its model level mutation score ($MS_{ML}(T)$) was calculated. At this stage the execution of the mutants was automated by use of USE (UML based Specification Environment) [10].

At the beginning of the second stage the UML/OCL models developed in the first stage were used to implement the systems in Java and the test suites for the systems were mapped into muJava test programs [15]. Next, for each system, its implementation was mutated by applying mutation operators defined for Java. The mutants were then executed against the test suites provided for the system and implementation level mutation score ($MS_{IL}(T)$) for each test suite T was calculated. This stage was fully automated due to the use of mujava [15].

Finally, for each test suite T the values of $MS_{ML}(T)$ and $MS_{IL}(T)$ were compared. Results of the experiment are presented and briefly discussed in Section III-D.

D. Experimental results and discussion

During the experiment each test suite T was assessed twice: first to get its model level mutation score ($MS_{ML}(T)$) and then to get its implementation level mutation score ($MS_{IL}(T)$). Table I shows the results of the experiment.

As it can be observed in Table I, for each test suite T the values of its $MS_{ML}(T)$ and $MS_{IL}(T)$ are very close to each other, in fact the results show that for each assessed test suite its model level mutation score differs from its implementation level mutation score by no more that 3%.

The results of the experiment let us to observe that:

- the model level mutation score of all test suites, but T4, was slightly (between 0.015 and 0.028) higher than their implementation level mutation score, and
- the difference between $MS_{ML}(T)$ and $MS_{ML}(T)$ for test suites that reached the model level mutation score over 0.80 remained at nearly constant level of about 0.02.

Both observations seem to suggest that the model level test suite assessment results may be seen as a reliable measurement of the test suite quality in general. The slightly higher fault detection rate observed for most of the test suites at the model level was to be expected, as the UML/OCL models do not define the processing of data as the implementations.

TABLE I
MUTATION SCORE FOR EXPERIMENTAL TEST SUITES

T	T1	T2	T3	T4	T5	T6
$MS_{ML}(T)$	0.78	0.87	0.90	0.76	0.83	0.84
$MS_{IL}(T)$	0.75	0.85	0.89	0.79	0.81	0.83

Nevertheless, the nearly constant difference between the model level and the implementation level results seems to indicate a regularity that would predict the quality of test suites based on the assessment results obtained at the model level alone.

The results obtained for test suites T1 and T4 shows that, for test suites attaining low model level mutation score, the test suite quality assessment performed at model level provides less predictable results than for the suites attaining higher model level mutation score. However, neither the overestimation of the quality of T1 nor the underestimation of the quality of T4 did not exceeded the 3% threshold. Moreover, a test designer, having developed a suite of such a low quality as T1 and T4, would most likely tend to improve it to achieve better score. Thus, it seems that the irregular behavior of T1 and T4 does not contradict the earlier conclusion regarding the reliability of model level test suite assessment results.

IV. Conclusions and Future Works

Mutation testing is an effective and reliable technique for assessing test suite quality with regard to their ability to detect faults specific to the given level, but a transferability of the assessment results between different levels of abstraction was not evaluated before. An experimental way to assess the relability of results obtained at model level was proposed in this paper. The results of the experiment let us to presume that for object-oriented systems, modeled in a form of UML/OCL class diagrams, the test suite's ability to reveal real faults can be reliably assessed at the model level. However, more experiments on larger systems should be carried out to verify the preliminary conclusions.

Future research concerning model level mutation testing should deal with the costs reduction problem. It seems that the migration to the model level alone lowers the number of generated and executed mutants, but other techniques should also be considered. The most efficient techniques should be the ones taking into account individual characteristics of modeled systems, such as proposed in [20], [21], [22].

Works on applying mutation testing at the model level should also include development of tools supporting generation of mutants. Availability of such tools would significantly increase the possibility of adapting such approach in practice.

References

[1] B. Aichernig and P. Salas, "Test case generation by ocl mutation and constraint solving," *in 5th International Conference on Quality Software,* Melbourne, 2005, pp. 64-71, http://dx.doi.org/10.1109/QSIC.2005.63.

[2] B. Aichernig, H. Brandl, E. Jobstl, W. Krenn, R. Schlick, S. Tiran, "Killing strategies for model-based mutation testing," *Software Testing, Verification and Reliability,* vol. 25(8), 2015, pp. 716-748, http://dx.doi.org/10.1002/stvr.1522.

[3] F. Belli, C. J. Budnik, A. Hollmann, T. Tuglular, W. E. Wong, "Model-based mutation testing - Approach and case studies," *Science of Computer Programming,* vol 120(1), 2016, pp. 25-48, http://dx.doi.org/10.1016/j.scico.2016.01.003.

[4] P. Black, V. Okun, Y. Yesha, "Mutation operators for specifications," *in 5th IEEE International Conference on Automated Software Engineering,* Grenoble, 2000, pp. 81-88, http://dx.doi.org/10.1109/ASE.2000.873653.

[5] A. Brucker, M. Krieger, B. Wolff, "A specification-based test case generation method for uml/ocl," *in International Conference on Models in Software Engineering,* Oslo, 2011, pp. 334-348, http://dx.doi.org/10.1007/978-3-642-21210-9-33.

[6] M. Daran and P. Thvenod-Fosse, "Software error analysis: A real case study involving real faults and mutations," *in ACM SIGSOFT international symposium on Software testing and analysis,* Mission Beach, CA, 1996, pp. 158-177, http://dx.doi.org/0.1145/229000.226313.

[7] R. A. DeMillo, R. J. Lipton, F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," *Computer,* vol. 11(4), 1878, pp. 34-41, http://dx.doi.org/10.1109/C-M.1978.218136.

[8] A. Derezinska, "Object-oriented mutation to asses the quality of tests, *in 29th Euromicro Conference,* Belek, 2003, pp. 417-420, http://dx.doi.org/10.1109/EURMIC.2003.1231626.

[9] T. Dinh-Trong, S. Ghosh, R. France, B. Baudry, F. Fleurey, "A taxonomy of faults for uml designs," *in 2nd MoDeVa workshop - Model design and ValidationModel Design and Validation Workshop,* Montego Bay, 2005.

[10] M. Gogolla, F. Buttner, M. Richters, "Use: A uml-based specification environment for validating uml and ocl," *Science of Computer Programming,* vol. 69, 2007, pp. 27-34, http://dx.doi.org/10.1016/j.scico.2007.01.013.

[11] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Transaction on Software Engineering,* vol. 37(5), 2011, pp. 649-678, http://dx.doi.org/10.1109/TSE.2010.62.

[12] Ying Jiang, Shan-Shan Hou, Jinhui Shan, Lu Zhang, Bing Xie, "An approach to testing black-box components using contract-based mutation," *Journal of Software Engineering and Knowledge Engineering,* vol. 18(1), 2008, pp. 93-117, http://dx.doi.org/10.1142/S0218194008003556

[13] R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes, G. Fraser, "Are Mutants a Valid Substitute for Real Faults in Software Testing?," *22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering,* Hong Kong, 2014, pp. 654-665, http://dx.doi.org/10.1145/2635868.2635929.

[14] W. Krenn, R. Schlick, S. Tiran, B. Aichernig, E. Jobstl, H. Brandl, "MoMut::UML Model-Based Mutation Testing for UML," *in 8th International Conference on Software Testing, Verification and Validation,* Graz, 2015, pp. 1-8, http://dx.doi.org/10.1109/ICST.2015.7102627

[15] Yu-Seung Ma, A. J. Offutt, Yong-Rae Kwon, "Mujava: An automated class mutation system," *Software Testing, Verification and Reliability,* vol. 15(2), 2005, 97-133, http://dx.doi.org/10.1002/stvr.v15:2.

[16] R. Schlick, W. Herzner, E. Jobstl, "Fault-based generation of test cases from uml-models approach and some experiences," *in 30th International Conference on Computer safety, Reliability, and Security,* Naples, 2001, pp. 270-283, http://dx.doi.org/10.1007/978-3-642-24270-0-20.

[17] J. Strug, "Classification of mutation operators applied to design models," *Key Engineering Materials,* vol. 572, 2014, pp. 539-542, http://dx.doi.org/10.4028/www.scientific.net/KEM.572.539.

[18] J. Strug, "Mutation testing approach to negative testing," *Journal of Engineering,* vol. 2016, 2016, http://dx.doi:10.1155/2016/6589140.

[19] J. Strug, "Mutation testing approach to evaluation of design models," *Key Engineering Materials,* vol. 572, 2014, pp. 543-546, http://dx.doi.org/10.4028/www.scientific.net/KEM.572.543.

[20] J. Strug J and B. Strug, "Machine learning approach in mutation testing," *Testing Software and Systems,* vol. 7641 of LNCS, Springer, 2012, pp. 200-214, http://dx.doi.org/10.1007/978-3-642-34691-0-15.

[21] J. Strug J and B. Strug, "Using structural similarity to classify tests in mutation testing," *Applied Mechanics and Materials,* vol. 378, 2013, pp. 546-551, http://dx.doi.org/10.4028/www.scientific.net/AMM.378.546 .

[22] J. Strug and B. Strug, "Classifying mutants with decomposition kernel," LNCS, Springer Berlin Heidelberg, vol. 9692, 2016, pp. 644-654, http://dx.doi.org/10.1007/978-3-319-39378-0-55

[23] M. Trakhtenbrot, "New mutations for evaluation of specification and implementation levels of adequacy in testing of statecharts models," *in Testing: Academic and Industrial Conference Practice and Research Techniques,* Windsor, 2007, pp. 151-160, http://dx.doi.org/10.1109/TAIC.PART.2007.23.

[24] S. Weissleder and B. H. Schlingloff, "Quality of automatically generated test cases based on ocl expressions," *in 1st International Conference on Software Testing, Verification, and Validation,* Los Alamitos, 2008, pp. 517-520.

[25] M-F. Wendland, "Abstractions on Test Design Techniques," *in Federated Conference on Computer Science and Information Systems,* Warsaw, 2014, pp. 1575-1584, http://dx.doi.org/10.15439/2014F316.