Using Meta-model Coverage to Qualify Test Oracles

Olivier Finot, Jean-Marie Mottu, Gerson Sunyé, and Thomas Degueule

LUNAM Université - LINA CNRS UMR 6241 - University of Nantes
2, rue de la Houssinière, F-44322 Nantes Cedex, France
firstname.lastname@univ-nantes.fr

Abstract. The definition of oracles is a significant part of model transformation testing. The tester has to ensure their quality. Mutation analysis that can be used to qualify test oracles is an expensive task which is also dependent on the transformation under test's implementation. In this paper we propose to use the coverage of the transformation's output meta-model by the oracles as an alternative to mutation analysis. This approach has been implemented and validated through experiments.

Keywords: Test, Oracle Quality, Meta-model Coverage

1 Introduction

Model transformations are among the key elements of Model-Driven Engineering. As for any other piece of software, developers must ensure that an implementation conforms to the specification. Software testing is a well-known technique for ensuring the correctness of an implementation. Testing a transformation consists of providing a set of test cases, where each test case is mainly composed of a test model and a test oracle. The role of the latter is to ensure that the output model produced by executing the transformation under test (TUT) over the test model is correct w.r.t. the transformation specification. Defining a test oracle is usually a manual task, performed by the tester, which relies on the specification.

In the context of model transformation testing, the qualification and generation of test models have already been studied [1]. However, beyond the scope of model transformation testing, few studies have been dedicated to the evaluation of test oracles. Mutation analysis [2] is a technique used to qualify test cases, according to their ability to detect real faults in an implementation. Faulty versions of the program, or mutants, are created by voluntary injecting faults in the TUT. To qualify test oracles, the mutants are then executed over the test data and the oracles check the produced models. The more mutants an oracle detects, the more efficient it is. However performing mutation analysis is an expensive task, mainly due to the manual creation of the mutants.

The goal of this paper is to propose a lighter approach to qualify oracles for model transformation testing. Instead of using mutation analysis, we rely on output meta-model coverage, i.e., we measure the elements of the output

meta-model that are exercised by the oracle. We claim that between two sets of oracles, the one covering more elements of the output meta-model is able to detect more faults than the other. When measuring the coverage of the output meta-model, the tester obtains two results: a coverage rate and a list of uncovered elements. The coverage rate is an indicator of the oracle's quality; the more an oracle covers the output meta-model, the higher its quality. The list of uncovered elements is an indicator on how to improve the oracle. To validate our approach, we developed a tool to qualify oracles, relying on meta-model coverage and we compared its results with the results of mutation analysis on two case studies. Experiments show that sets of oracles with a higher coverage rate kill more mutants, and therefore detect more errors.

2 Context

Test case quality has been studied both in the general scope of software testing and in the particular topic of model transformation testing.

2.1 Test Case Representativeness

Test Data Representativeness Several studies have been conducted on the generation or selection of test data. Many criteria to evaluate test data rely on the coverage of the System Under Test (SUT) by the test data. Zhu et al. [3] list several of these criteria. In the simplest criterion, the test data must cover each of the SUT's instructions. In another one, the tester checks that all the execution paths are covered during the execution of the SUT over the test suite.

Other approaches rely on the SUT's specification rather than it's implementation. With *Model-Based Testing (MBT)*, the tester generates test cases using a model specifying the SUT. Utting et al. [4] classified existing approaches and tools using MBT. In their study, the test case qualification relies only on the test data selection criterion. This selection can be based on the coverage of the model or it can also be random and stochastic. Also in the work of Dias Neto et al. [5], only the criteria of control or data flow coverage are presented on the topic of test case evaluation.

Test Oracle Evaluation In a test case, the oracle is as important as the test data. However, while the generation and selection of test data has been the subject of many studies, there has been fewer about the oracle. Bertolino [6] recognized that the oracle quality is a topic that should be studied more. Staats et al. [7] also insist on the importance of the test oracle. They propose to describe a testing system as a collection $(P, S, T, O, corr, corr_t)$ where:

- S is a set of specifications
- P is a set of programs
- T is a set of test data
- O is a set of oracles working as predicates

- $corr \subseteq P \times S$
- $corr_t \subseteq T \times P \times S$

O is a set of oracles (predicates) o such that $o : T \times P \rightarrow Boolean$. For a given $t \in T$ and $p \in P$, $o(t, p) = true$ denotes that the test passes. $corr(p, s) = true$ denotes that p is correct w.r.t. s. $corr_t(t, p, s) = true$ denotes that the specification s holds while executing p with the test data t. Hoffman [8] considers completeness as one characteristic of test oracles. Completeness can go from no prediction of the expected result to a complete duplication that is another implementation of the SUT. Staats et al. [7] also mention the oracle completeness but they define it differently; an oracle is complete for a program p and a specification s if for each test data t:

$$corr_t(t, p, s) \Rightarrow o(t, p)^1$$

If p is correct according to s for test data t, then the test passes. A complete oracle does not produce false positives. They also consider and define the soundness of an oracle; an oracle is sound if:

$$o(t, p) \Rightarrow corr_t(t, p, s)$$

If the test passes, then p is correct w.r.t. s for the test data t. Therefore, if there is an error in the program the test fails. An oracle that is both sound and complete is perfect:

$$\forall t, o(t, p) \Leftrightarrow corr_t(t, p, s)$$

Staats et al. also propose to compare test oracles based on their power. They state that an oracle o_1 is more powerful than an other o_2 for a test set TS (written $o_1 \geq_{TS} o_2$) for a program p and a specification s if:

$$\forall t \in TS, o_1(t, p) \Rightarrow o_2(t, p)$$

For a given test case, if o_1 passes then o_2 passes as well. If o_2 detects a fault then do does o_1. However, if o_2 passes then o_1, which is more powerful might detect a fault. Mutation analysis can then be used to qualify test oracles [9,10]. In this case it evaluates an oracle's ability to detect faults; therefore it could be used to compare test oracles. The more faults an oracle detects, the more powerful it is. We further discuss the use of mutation analysis to qualify test oracles and its drawbacks in the next section.

2.2 Test Case Quality in Model Transformation Testing

In this section we discuss published studies on the topic of test case quality for model transformation testing. They cover the generation and selection of representative test models or the evaluation of the oracle.

[1] For a predicate $Q(x)$, we simply write $Q(x)$ instead of $Q(x) = true$

Test Models Generation Fleurey et al. [1] propose to qualify test models based on their coverage of the input meta-model. They adapted criteria defined by Andrews et al. [11] for UML Class diagram coverage:

- `Class Coverage`: each meta-class is instantiated at least once.
- `Association End Multiplicities`: for each association extremity, each representative multiplicity must be covered.
- `Class Attribute`: for each attribute, each representative value interesting for the tester must be covered.

The last two criteria use the notion of representative value. The representative values are determined using partition analysis. An attribute or a multiplicity's value domain is partitioned into equivalence classes inside which the transformation's behavior is believed to be the same. One value is chosen in each equivalence class. Sen et al. [12] developed a tool based on these criteria to automatically generate test models.

Oracle Qualification Mottu et al. [13] use mutation analysis to evaluate model transformation test oracles. The process is divided into two steps: one to get qualified test models, one to get qualified test oracles.

In the first step, mutants of the TUT are created by each time adding a single fault. Each mutant is then executed over the test models. The output models produced by the mutant are compared to the ones produced by the TUT. If there is a difference, the mutant is killed. The mutation score is the ratio of the number of killed mutants over the total number of mutants. The higher the mutation score, the better the test models.

In the second step, the test oracles are used to kill the mutants. Since the qualified test models kill the mutants, we know their faults are detectable and we evaluate the capacity of the oracles to detect them. Another mutation score measures the number of mutants killed by the test oracles. Similarly to the test models, the higher the mutation score, the better the oracle.

However, despite its effectiveness, mutation analysis has two main drawbacks. The first one is that the application of mutation analysis depends on the TUT's transformation language. Mottu et al. defined language independent mutation operators, but the creation of mutants is manually done by the tester and the application of the operators depends of the transformation language. The second one is that mutation analysis is an expensive task due to the costs of execution and lack of automation.

3 Using Meta-model Coverage to Compare Test Oracles

Our goal is to provide an alternative to the expensive and language-dependent mutation analysis process, qualifying test oracles dedicated to model transformations. To qualify an oracle we measure its coverage of the TUT's specification. Our approach focuses on the models and is independent from the TUT and its

implementation language. It can be applied to any model in the widely used EMF framework. We first present our approach, then detail the process of measuring the coverage of the specification by the oracles.

3.1 Qualifying Model Transformations' Test Oracles

Staats et al. [7] define the power of an oracle as its ability to detect faults. This ability is evaluated thanks to mutation analysis: an oracle killing more mutants than another one is better. We propose qualifying oracles by measuring their coverage of the output meta-model. Instead of using mutation analysis to qualify any oracle in any test suite, we use it to validate our proposal Section 4.

When applying our approach to a transformation and a set of oracles (for instance, expected output models for each of the test models), the tester will measure the coverage of the output meta-model by all of these oracles together. The result of this measure is composed of two elements: a coverage rate and a list of uncovered elements. The goal of the tester is to maximize the coverage in order to obtain an efficient oracle set. If the coverage rate is lower than 100%, the oracles may have room for improvement. To improve the oracle, she will try to cover one or more of the uncovered elements.

However, reaching a coverage rate of 100% is not always feasible. The output meta-model may contain elements that should not be instantiated in a correct output model according to the transformation's specification. Given the list of uncovered elements, the tester may decide that some of these elements should not be covered. Therefore, the tester does not measure the coverage based on the complete output meta-model. She starts by defining the *effective* output meta-model, containing only elements handled by the transformation, according to its specification.

3.2 Output Meta-Model Coverage

We measure the coverage of the output meta-model by a set of test oracles. Fleurey et al. [1] qualify test models based on their coverage of the input meta-model. We propose to adapt these criteria for the coverage of the output meta-model. We use the same criteria for the coverage of the meta-model (`Class Coverage, Association Coverage, Class Attribute`). However, while Fleurey et al. combine these criteria to build representative models, we do not.

For input models, it is important to combine elements' values as different combinations can produce several different results. However, for output models it is not as relevant. Fleurey et al. proposed a way to handle inheritance links, only considering concrete meta-classes. For each of them, they check all their proper and inherited properties (attributes and references). We use the same process. When measuring the meta-model coverage of a given model, we mark as covered any EAttribute or EReference instantiated in the model. Then, if all of its properties are covered each concrete EClass is marked as covered.

Measuring the coverage of an output meta-model by a set of oracles is a two step process (see Figure 1):

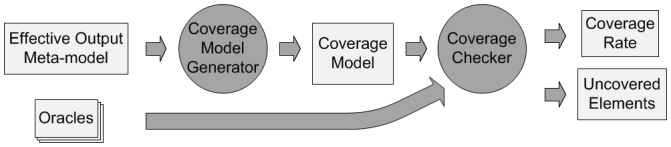


Fig. 1. Measuring Output Meta-model Coverage of Test Oracles

1. The first step transforms the meta-model MM_{out} to add coverage information. Each element to be covered is annotated. This annotated meta-model is the coverage model.
2. In the second step, the Coverage Checker analyses the coverage model and a set of oracles, producing two outputs. These outputs are the computed coverage rate and a list of all the uncovered elements.

4 Experiments and Discussion

In this section, we validate our approach on two case studies. First we present the two transformations under test (already used in previous works as detailed in [14]), then we detail our testing protocol and finally we discuss the results we obtain (experimental material is available[2]).

4.1 Case Studies

1. **Flattening of a Finite State Machine** `fsm2ffsm` flattens a hierarchical state machine. The input model of the transformation is a hierarchical state machine, the output model is another state machine expressing the same behavior without any composite state. This transformation has been implemented in Kermeta[3].
2. **UML to CSP** `uml2csp` transforms a simplified UML activity diagram into its corresponding modeled CSP program. We implemented it in ATL[4].

4.2 Evaluation Protocol

We use mutation analysis to evaluate the quality of the oracles in the same way as Mottu et al. in [13]. First, we create a set of mutants. Second, we create a set of test models which reaches a 100% mutation score, meaning that those test models are able to highlight the injected faults. Third, we create a set of oracles and we measure how many injected faults they detect (knowing that test models highlight all of them).

[2] http://pagesperso.lina.univ-nantes.fr/%7Emottu-jm/development-en.html
[3] http://www.kermeta.org
[4] http://www.eclipse.org/atl

Mutation Analysis Mottu et al. [13] proposed language independent mutation operators corresponding to faults that can occur in a model transformation's implementation. We create our mutants by applying these operators to our case studies implementations. Once the equivalent mutants are removed, we obtain 83 mutants for `fsm2ffsm` and 137 for `uml2csp`.

Test Models We combine input meta-model coverage and testing knowledge to create test models reaching a 100% mutation score. Using the approach proposed by Fleurey et al. [1], we define model fragments. The intent of these fragments is to cover the input meta-model according to a strategy. Additionally, we improve the fragments using our own knowledge of the transformation [12], increasing the coverage of the specification.

For `fsm2ffsm`, we obtain 11 fragments using the IFCombΣ strategy [1]. Each model fragment is completed producing two test models of different sizes. We finally obtain 22 test models.

For `uml2csp`, the IFCombΠ strategy [1] is used to obtain 8 model fragments. The low number of fragments is due to the fact that there are few references and many inheritances in the output meta-model used for this transformation, and almost none of the inherited classes or their superclasses contain attributes or references. Thus, in order to cover these classes, for each reference towards the superclass, we define one model for each of its non abstract inherited classes. We define all the possible combinations, and after eliminating the incorrect cases, we create 35 test models.

Test Oracles For each case study we create two sets of oracles using the two main existing oracle functions [14]: `set1` uses comparison with an expected output model while `set2` uses contracts.

The oracles from `set1` rely on expected output models that are compared to the produced ones. To validate our approach we need several subsets of oracles covering differently the effective output meta-model. Therefore we start by creating partial oracles controlling only part of the produced output models, as introduced in previous work [14]; then we increase the meta-model coverage of the oracles. The partial oracles are systematically created by ignoring successively each meta-class, each attribute, and each reference. In each partial oracle, we remove only one element.

The oracles from `set2` use contracts expressing properties on the output models, or between the test models and their corresponding outputs, as used by Cariou et al [15]. We write small contracts. Each contract corresponds to a property we want to check; for instance one contract checks that there is no composite states in the result of the transformation that flattens a state machine. Those contracts cover differently the meta-model. In addition, by incrementally combining our small contracts we build new ones improving the coverage of the output meta-model. We start by choosing the contract with the lower coverage of the output meta-model, then we gradually add each of the others. Each time we add the contract which will minimize the coverage rate's improvement.

Afterwards, for each oracle subset, we measure its coverage of the effective output meta-model.

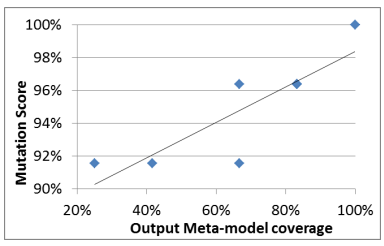
4.3 Results and Discussions

To validate our approach we must answer the following questions:

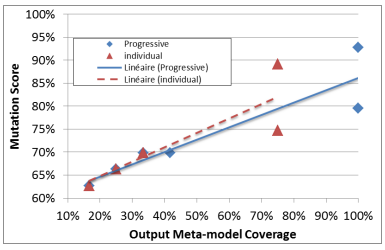
1. Can we measure the coverage of the output meta-model by a test oracle?
2. Is the coverage of the output meta-model related to the mutation score?
3. Does a 100% coverage rate indicate that the set of oracles is sound for the transformation?
4. Is the coverage of the output meta-model a good indicator of the quality of a set of oracles?

We developed and used tools to measure the coverage of a meta-model by a test oracle (or a set of test oracles) thus answering the first question.

In order to answer the second question, let us take a look at the graphs from Figure 2 and Figure 3. They show the relationship between meta-model coverage rate and mutation score, for each case study and for both types of oracles. On the left hand side graphs, each mark corresponds to a subset of oracles (or several subsets of oracles having the same coverage rate and the same mutation score). We also add a trend curve to see the impact of the coverage rate on the mutation score. The right hand side graphs display two series of values. Each triangle mark corresponds to one individual small contract, while each square mark represents one step of the definition of the incremental contracts. We add a trend curve for each of these two series, the dashed one corresponds to individual oracles and the solid one corresponds to the incremental ones. In the graph from Figure 2(a), we can see that by combining input meta-model coverage strategies with our knowledge of the transformation, we kill all of the mutants. We have 9 subsets of oracles, however 4 of them having the same coverage rate kill the same number of mutants. The subset of oracles with non partial expected models kills all of the mutants. In this case we can see that when the coverage rate increases then the mutation score does not decrease, at least, it stays unchanged. Moreover the



(a) Oracles using Expected Outputs



(b) Oracles using Contracts

Fig. 2. Relations Between Coverage and Mutation Score with fsm2ffsm

trend curve is strictly growing, meaning that the mutation score generally grows alongside the coverage rate. These two measures are related.

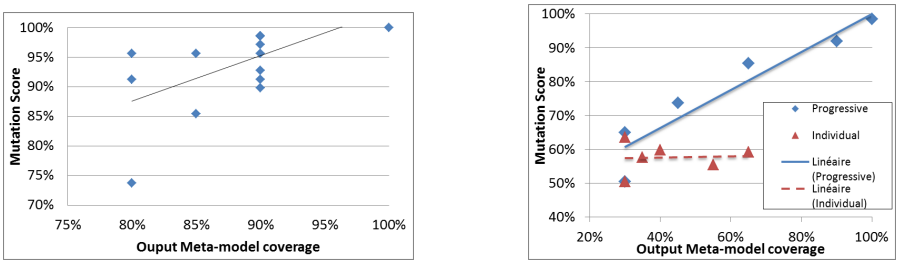
In the graph from Figure 2(b), we can see that with our contracts we kill 93% of the mutants. Both series contain 8 contracts and in both cases 2 pairs of contracts have the same coverage rate and mutation score. We can see from both trend curves that improving the coverage rate improves the mutation score.

Answering the third question we can see that while we cover the whole effective output meta-model, we do not kill all the mutants. Therefore a coverage rate of 100% does not imply that our oracles are sound.

For the second case study, in the graph from Figure 3(a) we can see the measures concerning the oracles relying on comparison with a partial expected output model. We have 16 subsets of oracles, however 5 of them have the same coverage rate along with the same mutation score. The subset of oracles with non partial expected models kills all the mutants. The other subsets of oracles have a lower coverage of the output meta-model and kill less mutants (between 73.72% and 98.54%). Answering the second question, similarly to the first case study, we see that the mutation score grows alongside the coverage rate.

As for the contracts, the graph from Figure 3(b) once again shows that we managed to entirely cover the effective output meta-model. Our final incremental oracle covering 100% of the output meta-model does not kill all the mutants, two of them are still alive. This result once again answers the third question. While the trend curve for the incremental contracts shows a noticeable improvement in the mutation score with the growth of the coverage rate, it is not so obvious for the individual partial contracts. Still in this case the trend curve indicates that improving the mutation score does not reduce the average mutation score. For both case studies, we are able to measure the coverage of the effective output meta-model by a set of oracles. From the obtained results, we notice that a better coverage rate comes with a better mutation score. However, we can also notice that entirely covering the effective output meta-model does not imply that the oracle is sound, since some faults may not be detected.

Still to answer our fourth question, while our approach is not as precise as mutation analysis, it has some advantages: it is lighter and it does not depend



(a) Oracles using Expected Outputs (b) Oracles using Contracts

Fig. 3. Relations Between Coverage and Mutation Score with UML2CSP

from the implementation of the transformation. Therefore, we can conclude that the coverage rate of the effective output meta-model by an oracle is a good indicator of the quality of this oracle.

5 Conclusion

In this paper we presented an approach to qualify oracles for model transformation testing, based on meta-model coverage. The more elements an oracle covers, the higher is its quality. We validated our approach by comparing the results of two different experiments with those of mutation analysis for the same experiments. The experiments showed that the oracles covering more meta-model elements killed more mutants. An important benefit of meta-model coverage over mutation analysis to qualify test oracles is that coverage is implementation independent and is less expensive to be performed.

References

1. F. Fleurey, B. Baudry, P.-A. Muller, and Y. L. Traon, "Qualifying input test data for model transformations," *Software and System Modeling*, vol. 8, no. 2, 2009.
2. R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," *Computer*, vol. 11, pp. 34–41, April 1978.
3. H. Zhu, P. A. V. Hall, and J. H. R. May, "Software unit test coverage and adequacy," *ACM Comput. Surv.*, vol. 29, no. 4, pp. 366–427, Dec. 1997.
4. M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing approaches," *Softw. Test., Verif. Reliab.*, vol. 22, no. 5, pp. 297–312, 2012.
5. A. C. Dias Neto, R. Subramanyan, M. Vieira, and G. H. Travassos, "A survey on model-based testing approaches: a systematic review," ser. WEASELTech '07.
6. A. Bertolino, "Software testing research: Achievements, challenges, dreams," in *FOSE*, 2007, pp. 85–103.
7. M. Staats, M. W. Whalen, and M. P. E. Heimdahl, "Programs, tests, and oracles: the foundations of testing revisited," in *ICSE*, 2011, pp. 391–400.
8. D. Hoffman, "A taxonomy for test oracles."
9. J.-M. Jézéquel, D. Deveaux, and Y. Le Traon, "Reliable objects: a lightweight approach applied to Java," *IEEE Software*, vol. 18, no. 4, pp. 76–83, 2001.
10. M. Staats, G. Gay, and M. P. E. Heimdahl, "Automated oracle creation support, or: How I learned to stop worrying about fault propagation and love mutation testing," in *ICSE*, 2012, pp. 870–880.
11. A. Andrews, R. France, S. Ghosh, and G. Craig, "Test adequacy criteria for UML design models," *Software Testing, Verification and Reliability*, vol. 13, no. 2, pp. 95–127, Apr. 2003.
12. S. Sen, J.-M. Mottu, M. Tisi, and J. Cabot, "Using models of partial knowledge to test model transformations," in *International Conference on Model Transformation*, Prague, Czech Republic, May 2012.
13. J.-M. Mottu, B. Baudry, and Y. Le Traon, "Reusable mda components: A testing-for-trust approach," in *proceedings of the MoDELS/UML 2006*, Oct. 2006.
14. O. Finot, J.-M. Mottu, G. Sunye, and C. Attiogbe, "Partial test oracle in model transformation testing," in *ICMT'13*, Budapest, Hungary, 2013.
15. E. Cariou, N. Belloir, F. Barbier, and N. Djemam, "Ocl contracts for the verification of model transformations," *ECEASST*, 2009.