# Using Meta-model Coverage to Qualify Test Oracles

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

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 oracle is a significant part of model transformation testing. The tester has to ensure their quality. Rather than using mutation analysis which is an implementation dependent and expensive task, we propose to qualify oracles by measuring their coverage of the transformation's specification.

**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 implementations are correct w.r.t. specifications. 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 test model over the transformation under test (TUT) 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. 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.

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 is. 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 an higher coverage rate kill more mutants, and therefore detect more errors.

This paper is organized as follows. Section 2 discusses the evaluation of test case quality, including outside the scope of model transformations. Section 3 details our approach to qualify oracles for model transformation testing based on their coverage of the output meta-model. Section 4 presents the experiments we ran on two case studies and discusses the results.

## 2 Context

Test case quality has been studied both in the general scope of software testing and in the 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] recognizes that the oracle quality is a topic that should be more studied. 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. $corr(p, o) = true$ 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 w.r.t. $p$ and $s$ if for each test data $t$:

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

If $p$ is correct w.r.t. $s$ 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:

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

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

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

If $o_1$ does not detect a fault for a given test case, then neither does $o_2$. Mutation analysis can then be used to qualify test oracles [9,10]. In this case it evaluates an oracle ability to detect faults; therefore it could be used to compare test oracles. The more faults an oracle detects, the more powerful it is. However, as we mentioned before, mutation analysis is too expensive to be used to compare oracles efficiently.

### 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 by 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. They create mutants for the TUT, then transform the input models with those mutants. In the following step, they compare the output models produced by the TUT with those produced by the mutants; if there is a difference, the mutant is killed. Among the alive mutants the tester must identify the ones that are equivalent to the TUT and remove them. Then, they evaluate the possibility of improving the mutation score by adding new test models. Once a sufficient mutation score is reached, they check the output models produced by the mutants with their oracles; the mutant is killed if the test fails. The higher is the mutation score, the better is 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 generic mutation operators, which are language independent. However, the creation of mutants is manually done by the tester and the application of the operators depends of the transformation language. Furthermore, mutation analysis is an expensive task due to the costs of execution and a 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, to qualify 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. 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

For Staats et al. [7] an oracle's power is its ability to detect faults. We use this power as an indicator of an oracle's quality. As with mutation analysis, an oracle of higher quality detects more faults. We qualify oracles by measuring their coverage of the TUT's specification. More precisely, we measure the coverage of the output meta-model by the set of oracles in the test suite.

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 oracles 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 w.r.t. 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, each concrete EClass is marked as covered if all of its properties are.

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

1. The first one 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.

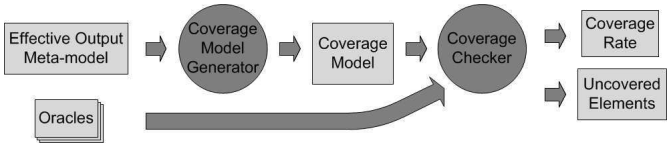



**Fig. 1.** Measuring Output Meta-model Coverage of Test Oracles

2. In the second one, 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` is a refactoring, the flattening of 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).

**Mutation Analysis** Using the mutation operators dedicated to model transformation testing in [13], we create mutants by injecting one single fault per version. Once the equivalent mutants are removed, we obtain 83 mutants for `fsm2ffsm` and 137 for `uml2csp`.

[2] `https://sites.google.com/site/qualifyingtestoraclesmt/`
[3] `http://www.kermeta.org`
[4] `http://www.eclipse.org/atl`

**Test Models** We combine input meta-model coverage and testing knowledge to create test models reaching 100% mutation score. Using the approach proposed by Sen et al. [15], 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 IFComb$\Sigma$ strategy. Each model fragment is completed producing two test models of different sizes. We finally obtain 22 test models.

For `uml2csp`, the IFComb$\Pi$ strategy 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 several sets of oracles, using two of the existing oracle functions [14]: comparison with an expected output model and contracts.

For the first oracle function we create a *complete expected model* for each test case. We also create several sets of *partial ones* as introduced in previous work [14]. All of our oracle sets for each case study rely on the same expected models, but for the partial ones we use a *filtered comparison* to control only part of the output model. We systematically create partial oracles by rejecting successively each meta-class, each attribute, and each reference. In each partial oracle, we remove only one element.

The other oracle function uses contracts expressing properties on the output models, or between the test models and their corresponding outputs, as used by Cariou et al [16]. 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. By incrementally combining our small contracts we build new ones. 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 set, we measure its coverage of the effective output meta-model.

### 4.3 Results and Discussions

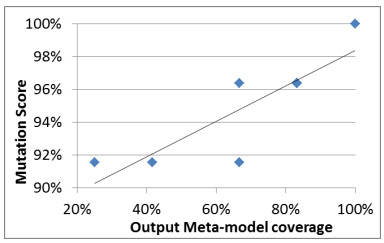
To validate our approach we must answer the following questions :

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?

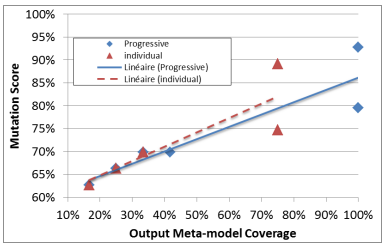
3. Does a 100% coverage rate indicates that the set of oracles is sound for the transformation?
4. Is the coverage of the output meta-model a good indicator of the quality of a set of oracles?

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 set of oracles (or several sets 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 dotted one corresponds to individual oracles and the full one corresponds to the incremental ones.




(a) Oracles using Expected Outputs




(b) Oracles using Contracts

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

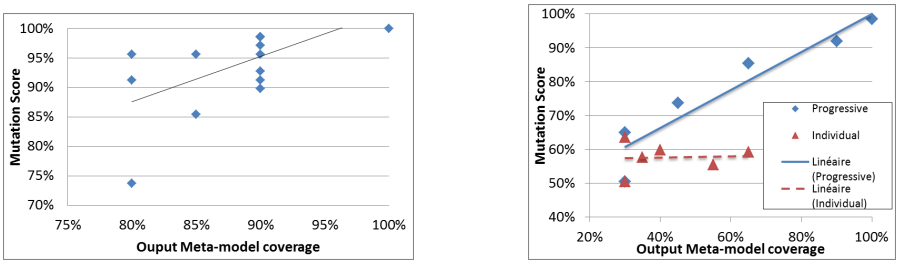
In the graph from Figure 2(a), we can see that by combining input meta-model coverage strategies with our knowledge of the transformation, we kill all of the mutants. We have 9 sets of oracles, however 4 of them having the same coverage rate kill the same number of mutants. The set of oracles without any filtering kill 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 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 incremental 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 our other 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 sets of oracles, however 5 of them have the same coverage rate along with the same mutation score. The set of oracles without any filtering once again kills all the mutants. The other sets of oracles have a lower coverage of the output meta-model and kill less mutants (between 86.86% and 95.91%). 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 diminish the average mutation score.




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

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

For both case studies, we are able to measure the coverage of the effective output meta-model by an oracle or 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 is not dependent 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 perform.

## 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. S. Sen, B. Baudry, and J.-M. Mottu, "Automatic model generation strategies for model transformation testing," in *ICMT'09.*, Zurich, Switzerland, Jul. 2009.
16. E. Cariou, N. Belloir, F. Barbier, and N. Djemam, "Ocl contracts for the verification of model transformations," *ECEASST*, 2009.