Test Cases Generation for Model Transformations from Structural Information

Wenquan Wang[1], Marouane Kessentini[2], and Wei Jiang[1]
[1]Missouri University of Science and Technology, USA
{wangw,wjiang}@mst.edu
[2]University of Michigan, USA
marouane@umich.edu

Abstract. Most of existing approaches for test cases generation to transformation mechanisms use a main criterion which is the coverage of source and target meta-model elements. However, this criterion is not sufficient in a real-world scenario. In fact, test-cases generated to cover meta-model elements cannot detect some transformation errors due to model-scalability reasons. These generated test cases are simple and different, in general, from source models that are used in an industrial setting. To make the situation worse, source models cannot be provided by industrial companies due to security/confidentiality reasons. Instead of real data (source models), corporations can provide structural information about their source models (e.g. number of classes, number of relationships, etc.). We propose a search-based approach for generating test cases based on the coverage of structural information in addition to meta-models coverage. The validation results on a transformation mechanism used by an industrial partner confirm the effectiveness of our approach.

Keywords- Search-based Software Engineering; testing; model transformation

1. Introduction

Model-driven engineering (MDE) is increasingly adopted in industry for being a new paradigm helping software developers to manage the growing complexity of systems being designed and implemented. In MDE, software models constitute the central artifacts in the software life cycle, going beyond their traditional use for automatically generating executable software. In fact, MDE aims to provide automated support for the creation, refactoring, and transformation of software models 24.

Although MDE is a promising approach to automated models transformation, so far it lacks techniques and tools for validating model transformations. One of the efficient techniques proposed recently is model transformation testing 14.Model transformation testing consists of generating a large number of different source models as test cases, running the transformation mechanism on them, and verifying the result using an oracle function such as a comparison with an expected results. Thus, model transformations testing includes with two challenging steps : the efficient generation of test cases and the definition of the oracle function. In this paper, we focus on the first step.

The generating of test cases for model transformation mechanisms is challenging. As explained in 5, testing model transformation is distinct from testing traditional implementations: the input data are models that are complex when compared to simple data types which complicate the generation and evaluation of test cases 6. These test

cases should be in general conformed to specific source metamodel such UML, database, etc. The main criteria used by existing work to evaluate test cases are the coverage of metamodel elements and reducing the number of test cases 789. However, these criteria are sometimes not enough to ensure the generation of efficient test cases. Simple test cases are generated to cover metamodel elements which are completely different from the real used data (source models to be transformed/ company's projects). When executing transformation mechanisms on industrial data many scalability issues can be detected due to some complex model fragments to transform or the huge number of transformation possibilities. One can use directly the real/industrial source models to test transformation rules. However, most of industrial companies, such as banks, do not accept to share their source models due to some security/confidentiality reasons.

To address these issues, we start from the observation that structural information, such as quality metrics, characterizing the source models can be collected from the industrial partner since it did not contains confidential information (e.g. name of model elements, information about customers, etc.). In this paper, we extend existing work by adding a new objective to maximize which the coverage of structural information to the real source models that will be transformed. To formally define the structural information, we used a set of metrics that characterize the proprieties of metamodel elements. Thus, in order to maximize the closeness to the real source models to be transformed, we minimize the distance between the metrics values of test cases and those of expected data to be transformed. To this end, we use a mono-objective optimization algorithm to generate test cases that maximize the coverage of metamodel elements while minimizing the structural-distance and the number of test cases. The proposed algorithm is an adaptation of Genetic Algorithm (GA) 12. The GA aims to explore the huge search space of the source metamodel in order to find the best solution ensuring the satisfaction of the three objectives described previously. For this, a custom tool was developed to generate test cases for the known case of transforming UML class diagrams (CD) to relational schemas (RS). The data related to CD was provided by an industrial partner, where the goal is to define and test a transformation mechanism to migrate to database. Of course, the bank gives us the structural information and not the real source models.

The primary contributions of the paper can be summarized as follows:

- We present a new approach to generate test cases for model transformations. Our proposal generates efficient test cases that are similar to the expected real-world data to be transformed.
- We report the results of an evaluation of our approach on industrial data and promising results are obtained. In fact, we success to generate artificial source models that are similar to expected industrial data with a maximum coverage of the source metamodels.
- We report the comparison results of our approach with an existing work 5 where only one and/or two evaluation criteria are used to evaluate test cases.

The remainder of this paper is as follows: Section 2 presents the relevant background and the motivation for the presented work using a real-world scenario; Section 3 describes the heuristic search algorithm; an evaluation of the algorithm with industrial validation is explained and its results are discussed in Section 4; Section 5 is dedicated

to related work. Finally, concluding remarks and future work are provided in Section 6.

2. Test-cases Generation for Model Transformation Mechanisms

This section describes the principles that underlie the proposed method for model transformation testing. It starts by presenting the overview of our proposal to generate test cases from structural information. Then, we provide the details of the approach and our adaptation of the genetic algorithm to the model transformation testing problem. As showed in Figure 1, our approach can be divided into two important components: the input/output of the testing process, and the main algorithm. We describe these components next.

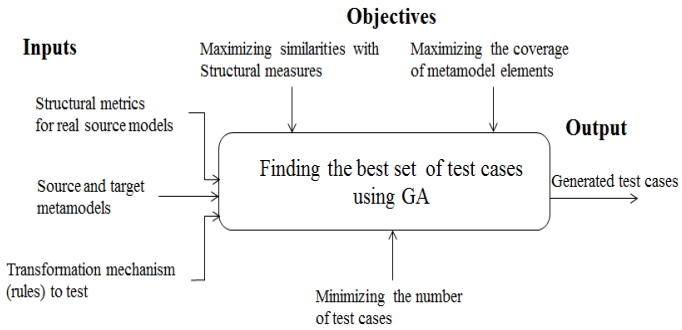

Fig. 1 Structural-based test cases generation overview

The Inputs of our approach are the source and target metamodels, the transformation mechanism (rules) to test, and structural metrics that characterize the company projects that will be transformed. The metamodels describe the source and target languages as described in Section 2. The transformation mechanism is a set of transformation rules mapping the different elements of source metamodel to their equivalent elements in the target metamodel such as class-to-table, attribute-to-column, etc. When defining a transformation mechanism some errors can be detected in the rules or the generated target models by executing a set of test cases. The third input of our approach is the metrics that provide useful information about the structure of projects to migrate. For class diagrams, we are using the most widely used metrics defined by Genero et al. [10]. These metrics include Number of associations (Naccoc): the total number of associations; Number of aggregations (Nagg): the total number of aggregation relationships; Number of dependencies (Ndep): the total number of dependency relationships; Number of generalizations (Ngen): the total number of generalisation relationships (each parent-child pair in a generalization relationship); Number of aggregations hierarchies (NAggH): the total number of aggregation hierarchies; Number of generalization hierarchies (NGenH): the total number of generalisation hierarchies; Maximum DIT (MaxDIT): the maximum of the DIT (Depth of Inheritance

Tree) values for each class in a class diagram. The DIT value for a class within a generalisation hierarchy is the longest path from the class to the root of the hierarchy; Number of attributes (NA): the total number of attributes; Number of methods (LOCMETHOD): the total number of methods; etc.
Using these inputs, the main goal (output) of our genetic algorithm is to generate artificial data (test cases) satisfying three objectives: maximizing the similarities with the expected metrics value, maximizing the coverage of metamodels elements, and minimizing the number of test cases. In this setting, a test case can be defined as any possible instantiation form the source metamodel.
As a huge number of test cases that can be generated to satisfy the three objectives, the test cases generation process is a seen as a combinatorial optimization problem. The number of possible solutions quickly becomes huge as the number of structural metrics and metamodel elements increases. A deterministic search is not practical in such cases, and the use of heuristic search is warranted. The search is guided by the quality of the solution according to the three described objectives. In the experiments, we tested different weights accorded to these objectives.
A high level view of our Genetic Algorithm approach to the test cases generation problem is introduced by Figure 2. As this figure shows, the algorithm takes as inputs the source and target metamodels, the transformation mechanism (rules) to test, and structural metrics that characterize the company projects that will be transformed. It finds a set of test cases (artificial source models) that best maximize the similarity with the structure of company-source models,maximize the coverage of metamodels, and minimize the number of test cases.

Input: Initial metamodels IMM
Input: Structural metrics STM
Input: Transformation mechanism (rules) TR
Output: Set of Test cases TC
1: I:= Instantiation(IMM)
2: Pop:= set_of(I)
3: initial_population(Pop, Max_size)
4: repeat
5: for all I ∈ Pop do
6: execute_test_cases(I, TR)
7: fitness(I) := compare(TR, STM)+ numberOfTestCases(TC) + (NumberOf elements(IMM) - coverage (IMM))
8: end for
9: best_solution := best_fitness(I);
10: Pop := generate_new_population(Pop)
11: it:=it+1;
12: until it=max_it
13: return best_solution

Fig. 2 Structural-based test cases generation overview

Lines 1–3 construct an initial GA population, which is a set of individuals that stand for possible test case solutions. Lines 4–13 encode the main GA loop, which explores the search space and constructs new individuals by combining model elements. For each

iteration, we evaluate the quality of each individual in the population, and save the individual having the best fitness (line 9). We generate a new population (pop+1) of individuals (line 10) by iteratively selecting pairs of parent individuals from population p and applying the crossover operator to them; each pair of parent individuals produces two children (new solutions). We include both the parent and child variants in the new population pop. Then we apply the mutation operator, with a probability score, for both parent and child to ensure the solution diversity; this produces the population for the next generation. The algorithm terminates when the termination criterion (maximum iteration number) is met, and returns the best set of test cases (best solution found during all iterations).

The encoding of an individual should be formalized as a mathematical function called fitness function. The fitness function quantifies the quality of the set of generated test cases. This function, to minimize, is based on three components and defined as:

$$f = n + (nbMMe - nbCovMMe) + \sum_{i=1}^{i=n} \sum_{j=1}^{j=m} \left| m_{i,j} - em_{i,j} \right|$$

Where *n* is the number of generated test cases; *nbMMe* is the number of metamodel elements; *nbCovMMe* is the number of covered metamodel elements; *m* is the number model elements in all test cases. Thus, the first objective is to minimize the number of test cases, the second objective is to maximize the coverage of metamodel elements (by minimizing the difference between the number of metamodel elements and the covered ones) and finally the third objective is to maximize the similarity with structural information of real source models to be transformed (by minimizing the difference between the metric values of test cases and real source models).

3. Experiments

To evaluate our approach, we conducted an experiment with industrial data. We start this section by presenting the data used and the types of transformation errors we considered in this study that should be detected using the generated test cases. Then, we report and discuss the obtained results.

In our experiments, we used structural information related to a set of class diagrams source models (CD) provided by an industrial partner acting in the banking sector. The efficiency of our generated test cases is evaluated using a precision score defined as follows:

$$\text{Precision} = \frac{\text{number of covered errors by the test cases}}{\text{number of errors}}$$

In fact, the important criterion to evaluate test cases is the coverage of the majority of expected errors. In addition, we compared the precision of our results with classical model testing approaches where mainly the coverage of metamodels is used. We presented in the next section the *average precision* (using different GA parameters) obtained using the coverage of structural information of the four class diagrams.

Finally, we evaluate the number of test cases required to cover all the expected errors.

Table 1. Class diagram statistics

Class diargram	#Elements
CD1	417
CD2	589
CD3	298
CD4	53

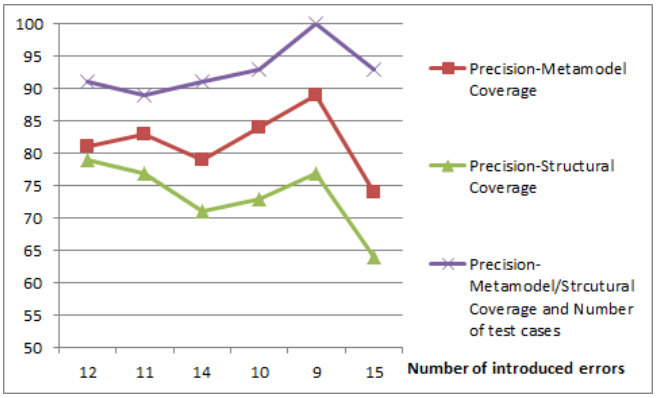

Fig 3. Precision results

As showed in Figure 3, the best precision scores are obtained when we combined metamodel coverage and strcutural informaiton coverage. In fact, for all the scenarios our proposal performs better than classical model-testing approach with a precision higher than 89%. Especially in the scenario where a high number of errors (15) are introduced in the tranformation mechanism to test. In this case the test cases generated using the combination of the three objectives cover more than 93% (14/15) of expected errors however using only one objective (metamodel coverage) only 64% of errors are covered by the generated test cases. We also investigated the types of transformation errors that were identified. As mentioned previously, the possible error sources were during specification of the model transformation mechanism: (i) the metamodels; (ii) the transformation logic (rules). Using our approach, all the error types were covered by the generated test cases. For instance, we obtained 100% precision in the scenario where 9 errors (3 rules-errors and 6 metamodel-coverage errors) are introduced. To conclude, the generated test cases using our proposal covered successfully most of expected errors much better than metamodel-coverage approach.

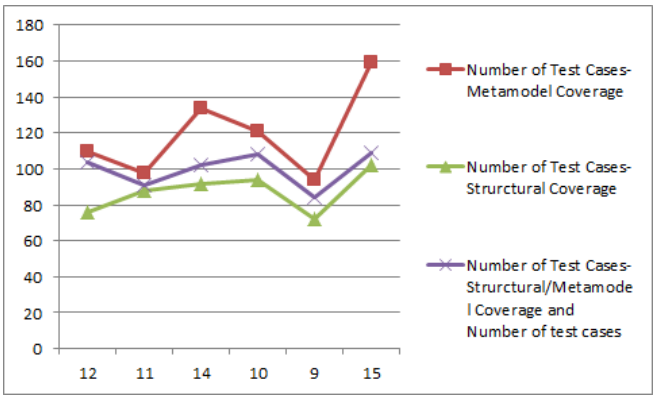

Fig 4. Number of generated test cases

An important consideration is the number of test cases used to cover the expected transformation errors. Since our approach takes into consideration the minimization of the number of generated test cases, Figure 4 shows that we obtained reasonable number of test cases that cover both structural information and metamodel elements. It is evident that low number of test cases is generated for structural information coverage since only real source models projects are expected to be covered. This is explains why few number of test cases are needed in this case. However, the coverage of metamodel elements requires the higher number of test cases in all the scenarios especially that minimizing the number of test cases is not considered as objective. Our proposal obtained lower test cases than the metamodel-coverage-technique with a better precision. For instance, in the scenario where 15 errors are introduced more than 160 test cases are generated by the metamodel-coverage technique. However, with our technique only 108 test cases are required. When we checked the results manually, we found that many redundant/similar test cases are generated when minimizing the number of test cases is not considered as an objective.

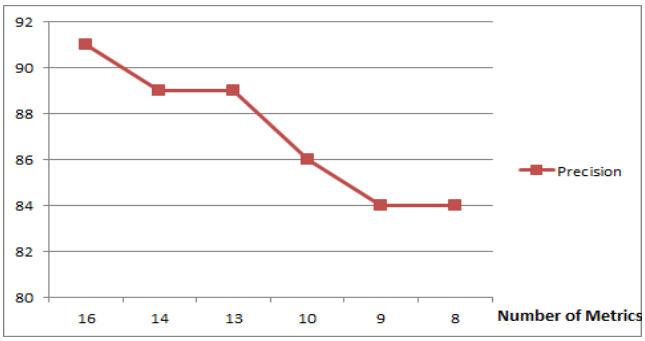

Fig 5. Precision vs. number of metrics used for structural information (scenario1=12 errors introduced)

Another important factor to evaluate when generating test cases from structural information is the number of structural metrics required to characterize the structure of

real source models to generate similar artificial data (test cases). Figure 5 shows the precision scores obtained on the first scenario (where 12 errors to cover are introduced) when we varied the number of structural metrics used. Our approach provides acceptable results (more than 80% as precision) when only 8 metrics are used thus we can conclude that the number of structural metrics required to collect from the industrial partner is not huge. This is important because our objective is to reduce the data required from the industrial partner as much as possible. However, the number of structural metrics depends on the used metamodels. We are planning to extend our validation in the future to covers other metamodels and then evaluating the impact of the structural metric information.

We executed our algorithm on a standard desktop computer and only a maximum of twenty minutes is required to generate the different results (with different GA parameters). Thus, our approach appears to be scalable from the performance standpoint. However, the execution time depends on the number of metrics used, number of metamodel elements and the number of rules used.

The precision results might vary depending on the test cases used, which are randomly generated, though guided by a metaheuristic. To ensure that our results are relatively stable, we compared the results of multiple executions for test cases generation using SA as showed in Figure 6. We consequently believe that our technique is stable, since the precision scores are approximately the same for seven different executions.

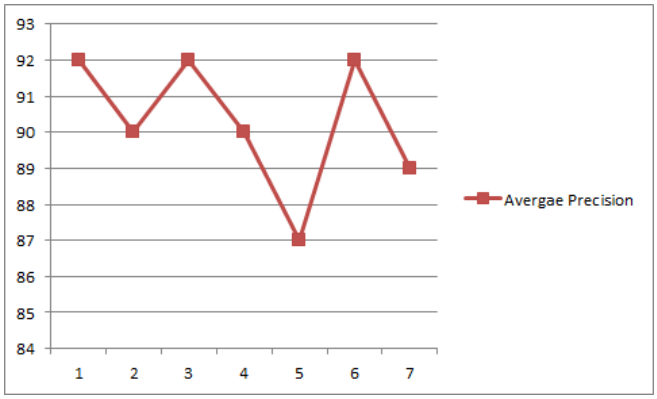

Fig 6. Stability results

4. CONCLUSION

In this paper, we described a new search-based for model transformation testing that take into consideration structural information of expected real source models to transform. The Inputs of our approach are the source and target metamodels, the transformation mechanism (rules) to test, and structural metrics that characterize the company projects that will be transformed. Using these inputs, the main goal (output) of our genetic algorithm is to generate artificial data (test cases) satisfying three

objectives: maximizing the similarities with the expected metrics value, maximizing the coverage of metamodels elements, and minimizing the number of test cases.
We evaluate our approach on industrial data and promising results are obtained. In fact, we success to generate artificial source models that are similar to expected industrial data with a maximum coverage of the source metamodels. Furthermore, we report the comparison results of our approach with an existing work where only one and/or two evaluation criteria are used to evaluate test cases.
Future work should validate our approach with more complex transformation mechanisms in order to conclude about the general applicability of our methodology. Also, in this paper, we only focused on the generation of test cases. We are planning to extend the approach by automating the detection of transformation errors. Furthermore, since we are considering many objectives to evaluate test case the use of multi-objective algorithms such as NSGA-II will be evaluated to find the best comprise. In addition, we will study the adaptation of the proposed approach to test object oriented code.

References

1. Mottu, J.M., Baudry, B., Traon, Y.L.: Model transformation testing: Oracle issue. In Proc. of ICST08.
2. Y. Lin, J. Zhang, and J. Gray. A Testing Framework for Model Transformations, in Model-driven Software Development. 2005, Springer.
3. Dimitrios S. Kolovos, Richard F. Paige, Fiona A.C. Polack. Model Comparison: A Foundation for Model Composition and Model Transformation Testing. In Proc. GaMMa, 2006.
4. Brottier, E., Fleurey, F., Steel, J., Baudry, B., and Traon, Y. L. Metamodel-based Test Generation for Model Transformations: an Algorithm and a Tool. In Proceedings of SSRE 2006
5. B. Baudry, F. Fleurey, J.-M. Jezequel, and Y. L. Traon. Automatic test cases optimization using a bacteriological adaptation model: Application to . net components. In ASE 2002.
6. Fleurey, F., J. Steel and B. Baudry, Validation in Model-Driven Engineering: Testing Model Transformations, In 15th IEEE International Symposium on Software Reliability Engineering. 2004
7. Andrews, R. France, S. Ghosh, and G. Craig. Test adequacy criteria for uml design models. Technical report, Computer Science Department, Colorado State University, 2006
8. Steel, J. and M. Lawley. Model-Based Test Driven Development of the Tefkat Model-Transformation Engine. In ISSRE'04, pp. 151-160, 2004. IEEE.
9. B. Baudry, T. Dinh-Trong, J.-M. Mottu, D. Simmonds, R. France, S. Ghosh, F. Fleurey, and Y. L. Traon. Model Transformation Testing Challenges. In IMDT workshop, 2006.
10. J. Kuster and M. Abd-El-Razik. Validation of model transformations- first experiences using a white box approach.In MoDeVa'06.
11. E. Cariou, R. Marvie, L. Seinturier, and L. Duchien. OCL for the Specification of Model Transformation Contracts. Proceedings of Workshop OCL and MDE, 2004.
12. D. E. Goldberg. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Pub Co, Jan 1989.

13. Varró, D., Pataricza, A.: Automated formal verification of model transformations. In: Jürjens, J., Rumpe, B., France, R., Fernandez, E.B. (eds.) CSDUML 2003: critical systems development in UML; proceedings of theUML'03 workshop, Technical Report, pp. 63–78. Technische Universität München, September 2003
14. Mottu, J.-M.,Baudry, B.,LeTraon,Y.: Mutation analysis testing for model transformations. In: Proceedings of ECMDA'06 (European Conference on Model Driven Architecture). Bilbao, Spain (2006)
15. Küster, J.M.: Definition and validation of model transformations. Softw. Syst. Model.5(3), 233–259 (2006)
16. Darabos, A., Pataricza, A., Varro, D.: Towards testing the implementation of graph transformations. In: Proceedings of GT-VMT Workshop Associated to ETAPS'06, pp. 69–80. Vienna, Austria (2006)
17. P. Sampath, A. C. Rajeev, S. Ramesh, and K. C. Shashidhar. Testing model-processing tools for embedded systems. In IEEE Real-Time and Embedded Technology and Applications Symposium, pages 203– 214, 2007.
18. M. Harman, The Current State and Future of Search Based Software Engineering, In Proceedings of the 29th International Conference on Software Engineering (ICSE 2007), 20-26 May, Minneapolis, USA (2007)
19. 41. A. Baresel, H. Sthamer, and M. Schmidt. Fitness function design to improve evolutionary structural testing. In GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, pages 1329–1336.
20. P. McMinn. Search-based software test data generation: A survey. Software Testing, Verification and Reliability, 14(2):105–156, June 2004.
21. Fleurey, F., Baudry, B., Muller, P.A., Traon, Y.: Qualifying input test data for model transformations. In: Software and Systems Modeling (2008)