Towards a model-driven engineering solution for language independent mutation testing

Pablo Gómez-Abajo[1], Esther Guerra[1], Juan de Lara[1], and Mercedes G. Merayo[2]

[1] Universidad Autónoma de Madrid (Spain)
[2] Universidad Complutense de Madrid (Spain)

Abstract. Mutation testing is a technique to assess test suite adequacy to distinguish between correct and incorrect programs. Mutation testing applies one or more small changes to a program to obtain variants called mutants. The adequacy of a test suite is measured by determining how many of the mutants it distinguishes from the original program. There are many works about mutation testing, but the existing approaches focus on a specific programming language, and usually, it is not easy to customize the set of mutation operators. In this paper, we present Wodel-Test, an extension of the Wodel tool that implements a language-independent mutation testing framework based on MDE principles.

Keywords: Model-driven engineering, domain-specific languages, model mutation, mutation testing, reverse engineering

1 Introduction

Mutation testing is an approach of software testing to assess the quality of test suites [4]. Mutation testing consists in injecting syntax changes in a program by using *mutation operators*. The mutations introduced into the original program aim at simulating common programming faults. The test suite's ability to detect faults in the generated mutants is measured by the *mutation score*, which is the percentage of mutants that the test suite *kills*, i.e., distinguishes from the original program.

Recent mutation testing frameworks are intended for a specific programming language. MiLu [7] is a specific mutation testing tool for C programs, while MuJava [10], MoMut [9], Major [8], Javalanche [11] and PITest [3] focus on Java. We found that Major is the only tool that provides a DSL that allows the definition of the mutation operators, but, as we point out, it only deals with Java programs. Thus, it would be very useful to create a language-independent mutation testing framework that allows designing and applying customized mutation operators and facilitates the creation of mutation testing environments. With this purpose, we have extended the development environment built for Wodel [5, 6].

Paper organization. First, Section 2 describes the process of our approach. Section 3 shows a running example, and finally, Section 4 ends with the conclusions and lines of future work.

2 Approach

First of all, we briefly introduce Wodel[3] [5], a DSL that facilitates the specification and creation of model mutations in a meta-model independent way. The language provides primitives for model mutation (e.g., creation, deletion, reference reversal), item selection strategies (e.g., random, specific, all), and composition of mutations. The Wodel environment allows creating Wodel programs, their compilation into Java, and the mutations execution. The Wodel tool can be extended with post-processor steps for particular applications, such as the automated generation of exercises [5], or the one presented in this paper.

We have provided the Wodel framework with a language-independent mutation testing extension called Wodel-Test. This approach uses model-driven engineering techniques, such as the aforementioned DSL for model mutation, as well as model-to-text, and text-to-model transformations. The process flow to create a Wodel-Test project is shown in Figure 1. We distinguish two kinds of users for our extension: the *testing plugin creators* and the *testers*.

Consequently, the process is separated in two stages. In the first stage, the testing plugin creation takes place. The user performing this action is in charge of designing the operators of interest for the selected domain. In addition, he/she must code the transformation of the projects to be tested into models, and viceversa; provide the code to compile the projects that are going to be tested; and finally, include the code for applying the test-suite and collecting the different results obtained from the mutation testing process. The result of this stage is a testing plugin that will be used by Wodel-Test for the selected domain.

In the second stage, the tester proceeds to create a Wodel-Test project using the previously generated plugin. This project includes the artifacts to be tested, and the test-suite to be applied. Finally, the tester executes the Wodel-Test project that will provide the mutation score corresponding to the test-suite. Currently, we have created proof-of-concept mutation testing frameworks for both Java and ATL.

3 Example

In this section, we illustrate Wodel-Test usage for the creation of a mutation testing framework for Java. We used the MoDisco [1] Eclipse plugin for manipulating Java projects and reverse engineering (i.e., converting Java programs into models conformant to a Java meta-model), and JUnit for the implementation of the test suite.

Firstly, we implemented some mutation operators using Wodel [2]. Next, we implemented the necessary methods to customize the testing environment for Java. In this case, we implemented the methods to transform Java projects into EMF XMI models, and viceversa; a method to compile Java code; and finally, a method to run the test-suite and collect the results of the mutation testing

[3] `http://gomezabajo.github.io/Wodel/`

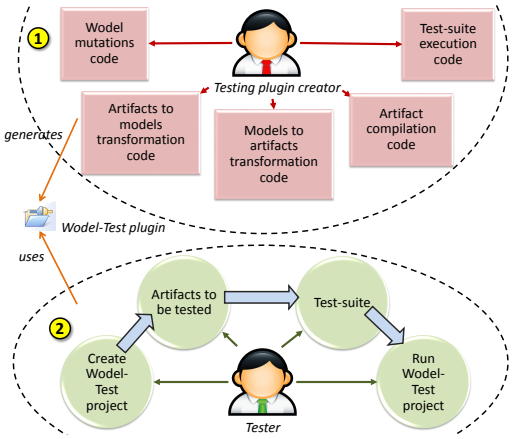

Fig. 1. Process of the WODEL-TEST mutation testing environment: 1. Creating the testing plugin. 2. Using the testing plugin

process. At this point, the plugin is ready, and can be used by the Java tester. Next, the Java tester launches the testing plugin from the Eclipse IDE, imports the projects to be tested, and the test-suite. Finally, the mutation testing process executes and shows the resulting mutation score.

Figure 2 illustrates this example[4]. The WODEL mutation replaces a `+` or a `-` operator by a `*` operator. The original Java program consists in a method to increment an integer value by 1. The testing plugin transforms this artifact into a model. Then, it applies the WODEL mutation and generates the mutants. In this example, the mutated Java program is the result of replacing the original `+` operator by a `*` operator. Next, the testing plugin transforms the generated mutant models into mutated Java programs. Finally, it executes the test-suite against the mutated programs, and shows the mutation score.

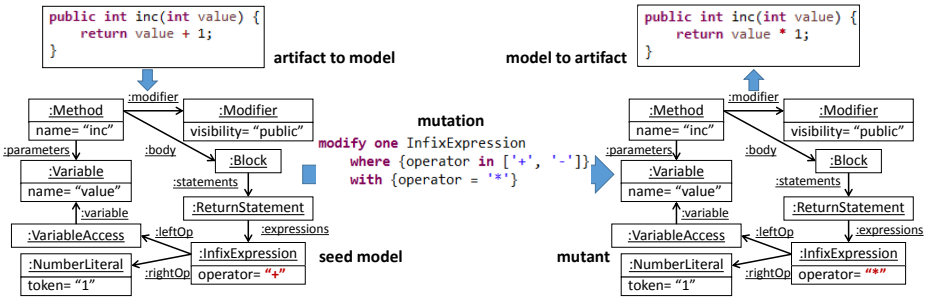

Fig. 2. WODEL-TEST example for Java

[4] A video showing this example can be found at: `https://youtu.be/2hx1Sy2uP34`

4 Conclusions and future work

In this paper, we have presented Wodel-Test, a model-driven engineering tool for mutation testing that is domain independent. We have described our approach and applied it to Java language. We have explained its benefits over existing tools, mainly its language-independence.

As a future work, we will extend Wodel-Test to show not only the mutation score corresponding to the test suite but also information about the quality of the operators and the level of difficulty to kill the generated mutants. In addition, we will include extensions to check that the mutants compile, and to identify and avoid duplicated and equivalent mutants. We will test our approach with other domains, and we will create specific Wodel mutation libraries for them.

Acknowledgements

Work partially funded by project FLEXOR (Spanish MINECO, TIN2014-52129-R), project DArDOS (Spanish MINECO/FEDER TIN2015-65845-C3-1-R) and the R&D programme of the Madrid Region (S2013/ICE-3006).

References

1. H. Brunelière, J. Cabot, G. Dupé, and F. Madiot. MoDisco: a Model Driven Reverse Engineering Framework. *Inf. and Software Techn.*, 56(8):1012–1032, 2014.
2. P. Chevalley and P. Thévenod-Fosse. A mutation analysis tool for Java programs. *International J. on Software Tools for Technology Transfer*, 5(1):90–103, 2003.
3. H. Coles, T. Laurent, C. Henard, M. Papadakis, and A. Ventresque. PIT: A practical mutation testing tool for Java (Demo). In *Proc.*, ISSTA 2016, pages 449–452, New York, NY, USA, 2016. ACM.
4. R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on test data selection: Help for the practicing programmer. *Computer*, 11(4):34–41, 1978.
5. P. Gómez-Abajo, E. Guerra, and J. de Lara. A domain-specific language for model mutation and its application to the automated generation of exercises. *Computer Languages, Systems & Structures*, 49:152 – 173, 2017.
6. P. Gómez-Abajo, E. Guerra, J. de Lara, and M. G. Merayo. A tool for domain-independent model mutation. *Science of Computer Programming*, In Press, 2018.
7. Y. Jia and M. Harman. MILU: A customizable, runtime-optimized higher order mutation testing tool for the full C language. In *Testing: Academic Industrial Conference - Practice and Research Techniques (taic part 2008)*, pages 94–98, 2008.
8. R. Just. The Major Mutation Framework: Efficient and Scalable Mutation Analysis for Java. In *Proc.*, ISSTA 2014, pages 433–436, New York, NY, USA, 2014. ACM.
9. W. Krenn, R. Schlick, S. Tiran, B. Aichernig, E. Jobstl, and H. Brandl. MoMut::UML Model-Based Mutation Testing for UML. In *Proc.*, ICST '15, pages 1–8, 2015.
10. Y.-S. Ma, J. Offutt, and Y.-R. Kwon. MuJava: A Mutation System for Java. In *Proc.*, ICSE '06, pages 827–830, New York, NY, USA, 2006. ACM.
11. D. Schuler and A. Zeller. Javalanche: efficient mutation testing for Java. In *ESEC/SIGSOFT FSE*, pages 297–298, New York, NY, USA, 2009. ACM.