Customizable Mutation Testing for Industrial Environments

Ali Parsai

University of Antwerp
Middelheimlaan 1
2020 Antwerpen, Belgium
ali.parsai@uantwerpen.be

Abstract

After the introduction of agile development techniques, the focus on testing has shifted more on the quality of the test suite. Consequently, the metrics adopted to evaluate such quality have become more important for both industry and academia. For the industry, it is a practical means to improve the effectiveness of the testing practices; for the academia, it is important to assess the validity of agile methods and comparison of different testing strategies. Mutation testing provides a repeatable and scientific approach to measure the quality of the test suite [4], and it is proven to simulate the faults realistically [1, 5]. Although the idea of mutation testing has been introduced in the late 1970s, it has not found widespread use in real scenarios due to its computationally expensive nature. Even though mutation testing is proven by academic research to be the strongest approach to quantify test suite quality, difficulties of this technique have caused simple code coverage metrics (e.g., statement and branch coverage) to become the de facto standard test suite quality metric in industry. Therefore to be practical in industrial settings, it is crucial to smoothen the integration of mutation testing in the industrial environment, and make use of the benefits of this technique [7]. However, this is not an easy task for four reasons. First, the overall process of building and testing the software is often complicated, leading to difficulties in the integration of mutation testing tools. Second, there is a lack of mature mutation testing tools and as consequence major build management systems have not developed common standards or requirements for such tools. Third, there is a lower demand for mutation testing due to a lack of background knowledge about the advantages it offers. For this reason, the development of new mutation testing tools is hindered, and their smooth integration is unlikely. Therefore, introduction of a tool capable of easy integration is an important step in popularizing the use of mutation testing.

To overcome the problem of integration, we propose a solution based on the Cha-Q[1] infrastructure[2] [3]. The Cha-Q environment aims to strike a balance between agility and reliability through change-centric quality assurance tools. This environment offers a first-class representation of software artifact changes. Making use of this infrastructure allows easy integration with compatible continuous integration tools, and easier transformation into new settings supported by Cha-Q infrastructure.

Our proposal is (i) to create a mutation testing tool on top of Cha-Q meta model, (ii) use ChaQeko/X (a tool under development in Cha-Q). ChaQeko/X is a program transformation tool based on Ekeko/X [2]. Its main characteristic is to be template-driven. We exploit this characteristic to provide customizable mutation operators, and adaptable to the needs of the developer.

[1] Change-centric Quality Assurance
[2] http://soft.vub.ac.be/chaq/

One important aspect of mutation testing is the use of mutation operators to generate mutants. These mutation operators are usually based on a simple fault model. However, this simplicity usually means that lots of *mutable statements* are found, and as a result lots of mutants are generated. This leads to adverse effects on performance of this procedure, and does not allow mutation testing to scale to larger software.

The use of simple mutation operators to generate the faults is justified based on two fundamental assumptions. First, the Competent Programmer Hypothesis states that a competent developer will only make mistakes that are solvable by making few syntactic changes [4]. Second, the Coupling Effect Hypothesis states that "complex mutants are coupled to simple mutants in such a way that a test data set that detects all simple mutants in a program will also detect a large percentage of the complex mutants" [6]. Even though both these hypotheses are true for simpler faults, the more complex the faults become, the more difficult it gets to detect and solve as them. Thus, there is a need for more complex fault models to be introduced to tackle this problem. Using ChaQeko/X allows us to give the developer the option to define their own mutation operators. These mutation operators can be more complex than the simple mutation operators that are being widely used. This also means that they will generate less mutants, and they can target types of faults that are relevant to the project itself rather than the generic faults.

References

[1] J. Andrews, L. Briand, and Y. Labiche. "Is mutation an appropriate tool for testing experiments? [software testing]". In: *Proceedings of 27th International Conference on Software Engineering, 2005 (ICSE 2005)*. 2005, pp. 402–411.

[2] C. De Roover and K. Inoue. "The Ekeko/X Program Transformation Tool". In: *Source Code Analysis and Manipulation (SCAM), 2014 IEEE 14th International Working Conference on*. 2014, pp. 53–58.

[3] C. De Roover, C. Scholliers, V. Jonckers, J. Prez, A. Murgia, and S. Demeyer. "The Implementation of the Cha-Q Meta-Model: A Comprehensive, Change-Centric Software Representation". In: *In Electronic Communications of the European Association of Software Science and Technology, Post-Proceedings of the 8th International Workshop on Software Quality and Maintainability (SQM14)* 65 (2014).

[4] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. "Hints on Test Data Selection: Help for the Practicing Programmer". In: *Computer* 11.4 (Apr. 1978), pp. 34–41.

[5] R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes, and G. Fraser. *Are Mutants a Valid Substitute for Real Faults in Software Testing?* Tech. rep. UW-CSE-14-02-02. University of Washington, 2014.

[6] A. J. Offutt. "Investigations of the Software Testing Coupling Effect". In: *ACM Trans. Softw. Eng. Methodol.* 1.1 (Jan. 1992), pp. 5–20.

[7] B. H. Smith and L. Williams. "On guiding the augmentation of an automated test suite via mutation analysis". In: *Empirical Software Engineering* 14.3 (2009), pp. 341–369.