Proteum/AJv2: A Mutation-based Testing Tool for Java and AspectJ Programs

Fábio Gomes Leme*, Fabiano Cutigi Ferrari†, José Carlos Maldonado*, Awais Rashid‡
*Computer Systems Department – University of São Paulo (USP) – São Carlos – Brazil
†Computing Department – Federal University of São Carlos (UFSCar) – São Carlos – Brazil
‡School of Computing & Communications – Lancaster University – Lancaster – United Kingdom
Email: filipegl@grad.icmc.usp.br, jcmaldon@icmc.usp.br
Email: fabiano@dc.ufscar.br
Email: marash@comp.lancs.ac.uk

Abstract—This paper describes the Proteum/AJv2 tool, which automates the application of AspectJ-specific mutation operators as well as unit mutation operators to both Java and AspectJ programs. This can be done through a newly created graphical user interface. The paper also describes the mutation process supported by Proteum/AJv2 and how the new interface facilitates the tasks required by mutation testing.

I. INTRODUCTION

Mutation testing [1] has shown to be an adequate test selection criterion that leads to the creation of test sets which are effective to reveal software faults. Furthermore, it is also adequate for the evaluation of existing test sets [2]. In this context, we herein describe the Proteum/AJv2 tool, which is an evolution of Proteum/AJ [3] that was conceived to support the mutation testing of AspectJ programs. Proteum/AJv2 evolves Proteum/AJ to support the mutation testing of both object-oriented (OO) and aspect-oriented (AO) programs. The contributions brought by Proteum/AJv2 are two-fold: (i) besides the AspectJ-specific mutation operators, Proteum/AJv2 now supports the application of traditional unit mutation to both Java and AspectJ programs; and (ii) Proteum/AJv2 brings a newly developed graphical user interface (GUI) that supports comprehensive test project management. To the best of our knowledge, Proteum/AJv2 is the only tool that automates the mutation testing of programs developed under these two paradigms. Thus, it represents a step towards the integrated mutation testing of OO and AO programs.

II. PROTEUM/AJV2’ ARCHITECTURE

Proteum/AJv2 partially implements a reference architecture for software testing tools called RefTEST [4]. RefTEST is based on separation of concerns (SoC) principles, on the Model-View-Controller (MVC) and three-tier architectural patterns, and on the ISO/IEC 12207 standard for Information Technology. Proteum/AJv2’s architecture is shown in Figure 1.

The Server Side includes the MVC modules. The alternative user interfaces appear on the Client Side. The core comprises the main concepts that should be handled by testing tools, as proposed in RefTEST [4]. In Proteum/AJv2, criterion maps to mutation testing, artefact maps to source code, requirement maps to mutant, and test case maps to test case itself. For example, the criterion module handles the mutant generation, compilation and analysis, while the test case module handles test case execution and evaluation.

Figure 2 shows a simplified UML diagram that illustrates some internal details of Proteum/AJv2’s mutation engine. The classes shadowed in grey represent additions implemented in the tool. Each mutation operator is encapsulated within a specific class. Basically, the highlighted classes have been introduced to implement the unit mutation operators for Java and AspectJ programs.

The newly introduced operators in Proteum/AJv2 are: CGCR, CLCR, CGSR, CLSR, VDTR, VTWD, OASN, OEBR, ORRN, SDWD, SMTL and SSDL. Such operators have been characterised as sufficient mutation operators due to their low costs (in terms of the number of produced mutants) and high effectiveness in revealing faults [5, 6]. Note that descriptions of the modifications performed by these operators can be found elsewhere [7]. Furthermore, modifications related to AspectJ-specific operators are described in our previous research [3], as well as external tools used for parsing source code files, pretty-printing mutants and database storage.

III. PROTEUM/AJV2’S GRAPHICAL USER INTERFACE

![Figure 1. Proteum/AJv2’s architecture based on RefTEST [4]](image-url)
**Proteum/AJv2** brings a newly developed graphical user interface (GUI). The set of requirements for the GUI was derived from existing testing tools such as **Proteum** [8] and **Jabuti** [9]. These requirements can be divided into: (i) test project management; (ii) test case handling (e.g. addition and visualisation); (iii) test requirement generation (that is, the mutants); and (iv) report generation. Design patterns such as Observer, Command and Singleton help **Proteum/AJv2** meet these requirements.

Figure 3 illustrates the **Proteum/AJv2**’s main window together with some of its embedded windows. To create a new test project, the user needs to first create a compressed file (in ZIP format) that includes the full source code of the system under testing (SUT). The compressed file must also include compilation and test execution directives, defined as Apache Ant tasks. Once the SUT is submitted to the tool, the GUI enables the test project customisation in several ways. For example, in the **Targets** window the user can choose the mutation operators to be applied to the selected elements of the SUT. In the **Source Codes** tab, the user can also choose the source files that will be tested using such operators. For each source file, the user can select specific internal targets (methods and types of advice) to which the unit mutation operators are applied. We highlight that this internal target selection feature was not available in the original **Proteum/AJ** [3].

The **Mutants** window exhibits the set of mutants for the current test project. The user can check details of each mutant such as the original and the mutated source files, the differences between these files, and the mutant status. The **Create** tab contains the options related to mutant generation (e.g. re-generating all mutants or generating mutants for newly selected mutation operators). The **Reports** window shows the summary of results for the current test project, while the **Test Cases** window (not shown in Figure 3) allows the execution of **JUnit** test cases.

**IV. Final Remarks**

This paper described the main features of the **Proteum/AJv2** tool that supports the mutation testing of Java and AspectJ programs. To the best of our knowledge, **Proteum/AJv2** is the first mutation testing tool that supports unit mutation testing of both Java and AspectJ programs, as well as AspectJ-specific mutations. As its predecessor [3], **Proteum/AJv2** leverages previous knowledge on developing testing tools [8, 9] and reference software architectures [4] in order to configure an integrated environment for testing Java and AspectJ applications. **Proteum/AJv2** is also expected to spread knowledge on software testing, in particular mutation testing, in the academic context. Consequently, it is also expected to support the technology transfer to industry, promoting the application of systematic testing in real-world projects.

We are currently defining a series of improvements in **Proteum/AJv2** to support experimentation. It includes creating parameterised scripts to automatically execute a chain of tasks such as test project creation, mutation generation, test case execution and coverage analysis. The design of customised reports is also included in the improvement plan.

**ACKNOWLEDGEMENTS**

This work was funded by FAPESP (grants 2011/21515-3 and 08/57870-9), CNPq (grant 573963/2008-8) and CAPES (grant 0653/07-1).

**REFERENCES**


