Liu, C.-C. et al. (Eds.) (2014). Proceedings of the 22nd International Conference on Computers in Education. Japan: Asia-Pacific Society for Computers in Education

On Using Mutation Testing for Teaching Programming to Novice Programmers

Rafael A. P. OLIVEIRA[a*], Lucas B. R. OLIVEIRA[ab], Bruno B. P. CAFEO[c] & Vinicius H. S. DURELLI[a]

[a]*Department of Computer Systems ICMC, University of Sao Paulo, Brasil*
[b]*IRISA Research Institute, University of South Brittany, France*
[c]*Opus Research Group, Informatics Department, PUC-Rio, Brasil*
*rpaes@icmc.usp.br

Abstract: In this paper we argue that the incorporation of experiences on testing activities, in particular mutation testing, in programming courses adds valuable knowledge to the learning process. Mutation testing is centered on the idea of creating test data for uncovering seeded faults in programs that slightly differ from the original program. These faulty programs are called mutants. Through source code analysis and test case execution, testers have to identify the differences between the original program and the mutants. To do so, testers must have a sound understanding of the program's control flow and instructions. This broad understanding of programming represents a key skill for novice students in programming courses. We evaluate the effects of using mutation testing to improve the learning process of novice students in programming courses. We conclude that the introduction of mutation testing in the learning process provides students with learning experiences that go beyond traditional lectures and hands-on programming courses.

Keywords: Programming, mutation testing, experimental study, computer-aided instruction

1. Introduction

This paper reports on an experience of using Mutation testing to support the learning process of novice students in an undergraduate programming course. Mutation testing (DeMillo, 1979) is a fault-based testing criterion which relies on typical mistakes programmers make during software development. Further, our research aims to use a software testing criterion to improve the learning experience of novice programmers. Hence, we present *Pascal Mutants*, a tool able to support mutation testing for Pascal programs. We have developed this tool specifically to support the learning processes in programming classes for undergraduate courses. We have chosen Pascal due to its block structures, statements and explicit variables declarations that make it a well-organized language for beginners. In addition, Pascal is a procedural language that has commands in a natural language from which developers can implement reliable and effective programs. Finally, we evaluate the effects of using mutation testing to improve the learning process of novice students in programming courses.

2. Pascal Mutants

The Pascal Mutants tool was designed to perform mutation at the unit level in Pascal programs. Through seven mutation operators, the tool injects artificial faults in a given original program, generating different faulty versions of the original program. Users can select specific mutant operators, create testing projects, manage and execute testing data, check the mutation score and compare original and mutant codes. Technically, Pascal Mutants integrates six main modules: (1) a Pascal syntax analyzer and syntax tree generator, (2) a compiler and loader of mutants, (3) a test case manager, (4) a results evaluator (test oracle), (5) a mutant manager, and (6) a test reporter.

Pascal Mutants has a syntax analyzer that groups tokens of a source program into grammatical production, generating a syntax tree that includes a tree-type representation of a source code written in

Pascal. To implement these elements we have followed the grammar specification suggested by Setzer and Melo (1981). Then, using the tool JavaCC (*Java Compiler Compiler*) and its pre-processor named JJTree, we have set an effective form to support the mutant creation through parsing of a Pascal code. After generating mutants, Pascal Mutants uses the resources of *GNU Pascal* to compile mutants and to create their associated binary files. During this process, depending on the mutation operator applied, some mutants may be automatically killed due to compilation errors, for instance. Pascal Mutants offers functionalities to receive and load test inputs, mark an equivalent mutant, and present test coverage and statistics

3. An Experimental Study

To conduct a preliminary assessment of the impacts of using mutation testing and the Pascal Mutants in programming courses, we have applied the concepts presented here in a framework of a four-hour course for a group of undergraduate students. We divided this experiment in two parts: (*Part I*) an experimental setup where we explained theoretical concepts of software testing (Experimental Setup); and (*Part II*) a *Controlled experiment* involving Pascal Mutants. Controlled experiments provide resources to compare more than one treatment to analyze outcomes (Wohlin et. al 2001).

3.1 Students

The group of subject students was composed of 20 undergraduate students. These students were enrolled in the second semester of the Bachelor of Computer Science course at UNESP (Universidade Estadual Paulista) campus Rio Claro, Sao Paulo, Brazil. All students had prior knowledge of about two-month of a course of algorithms and Pascal language, after which they were considered to be novice programmers. This activity was conducted during one of their first experiences on practicing theoretical concepts in a laboratory. They had no prior knowledge of either software engineering or testing concepts.

3.2 Experimental Setup (Part I)

In the first part of the course we asked the students to turn off their computers. Then, we presented general concepts of software testing, including theoretical concepts and examples. After that, we focused on the mutation testing criterion with examples and concepts. Next, we introduced the students to Pascal Mutants, presenting several examples of basic operations and how to explore all of the tool's functionalities. During this part of the experiment, the participants had no contact with the tool.

3.3 Conduction of the Controlled Experiment (Part II)

In the second part of the experience, we conducted a controlled experiment in the laboratory. This experiment has allowed us to systematically observe the attitudes of all participants. Aiming to solve the practical activities detailed in Section 4.6, we divided our subjects randomly in two groups of ten students each. Students in the first group (Group 1 – G1) were asked to conduct their activities using their preferred Pascal compiler. Then, G1, which was considered the *control group* of this experiment, was assigned to only use *ad-hoc* techniques and their own understanding throughout the experiment. After that, we designated the students allocated to the second group (Group 2 – G2) to use the Pascal Mutants tool to perform the same activity. Regarding G2, which is considered the *treatment group*, we helped the students to download and set Pascal Mutants on their computers. Besides that, we avoided giving different attentions for the groups to prevent possible biases.

3.4 Subject Program and Survey

After the setup, we provided a Pascal source code of a program that produces a sequence of numbers named the *Fibonacci series*. We have set a Pascal program to receive three parameters by command line: (1) the first element of a Fibonacci series (starting point), (2) the second element of a Fibonacci series, and (3) the limit of the series (ending point). Then, we have implemented a Fibonacci algorithm using these parameters that were supposed to be informed by the user.

We have measured the effects of our approach using a survey. In this survey, we did not provide any specifications about the program. Besides, the variable names offered no clues of their functionality. In this way, we intentionally avoided any understanding of what the program is intended to do. Since the students had no previous knowledge about the Fibonacci algorithm, it was expected that there would be different descriptions of the Fibonacci sequence. Besides the amount of time spent by the students, the survey was composed of five essay questions about the generic operation and functionalities of the subject program.

4. Results and Discussion

Results pointed out that Pascal mutant positively affects the understanding about the source code. The students included in G2 (using Pascal Mutants) had better scores than the students in G1 (using compilers). With the exception of the fourth question, students in G2 obtained a better mean score in all of the questions. Analyzing the responses, we noticed that in general, the students in G1 tried to solve the survey's questions through trial and errors approaches exploring the compiler with different inputs and observing the program behavior. On the contrary, students in G2 had to think more carefully about the algorithm to 'kill' mutants, consequently they were able to formulate more accurate answers about the program. Using the mutation analysis was a valuable experience for novice programmers.

Regarding the time analysis, we observed that students in G2 took a long time to finish their experiments. This reveals that the practical usage of a mutation testing tool instead of regular compilers may be considered a disadvantage. This is due to the fact that students in G1 had previously obtained skills to conduct the experiment using their preferred Pascal compiler. On the other hand, students in G2 had to dedicate more effort to understand all of the resources and functionalities provided by Pascal Mutants tool.

5. Conclusions

Traditional methods for teaching programming may not provide enough experiences to reach all students' expectations. This paper proposes and evaluates the usage of mutation testing criterion as a resource to improve the learning processes of novice students in programming courses. Pascal Mutants, an open source and intuitive testing tool for applying mutation testing in Pascal programs, is presented. We conducted an empirical analysis on a group of novice programmers represented by undergraduate students. Our findings show the feasibility of our approach and its benefits. The use of mutation testing concepts may provide a more complete and accurate understanding of the whole functioning of a program for novice programmers. Through a trade-off analysis of this approach, we highlighted the dependence of specific testing tools. We conclude that the incorporation of testing activities into programming courses generates resources that go beyond lectures and plain hands-on experience, contributing to the learning of novice programmers. In view of this, the present paper explores the idea of using artificial defects as a means of diagnosing real defects.

References

DeMillo, R. A., Lipton, R. J., and Sayward, F. G. (1979) Program mutation: A new approach to program testing, in *Infotech State of the Art Report, Software Testing*, pp. 107–126.

Setzer, V. W. and Melo, I. S. H. (1981). A Construção de um Compilador. (in portuguese) *Instituto de Matemática, Estatística e Ciência da Computação da UNICAMP*, Rio de Janeiro, Brazil: ed. Campus.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B. and Wesslén, A. (2001). Experimentation in software engineering: An introduction. *Kluwer: Academic Publishers*, Norwell, MA, USA.