Available online at www.sciencedirect.com

ScienceDirect

Procedia - Social and Behavioral Sciences 228 (2016) 215 – 221

Procedia Social and Behavioral Sciences

2nd International Conference on Higher Education Advances, HEAd´16, 21-23 June 2016, València, Spain

Benefits of a testing framework in undergraduate C programming courses

Dieter Pawelczak[a]*

[a]*Faculty of Electrical Engineering and Computer Science, Universitaet der Bundeswehr Muenchen, 85579 Neubiberg, Germany*

Abstract

We introduced a JUnit like testing framework in an automated grading system for C programming assignments in order to reduce on the one hand the failure/ dropout rate of the course and on the other hand to master rising enrollments in the lab course. The testing framework is integrated into the Virtual-C IDE; a programming environment especially designed for learning and teaching the C programming language. In contrast to the previous system design, our new system provides detailed information on test results to the students. In order to prevent coding against the tests instead of coding according to the specification a high test coverage and randomized test data are used. The paper presents the results of a students' evaluation of the course with two different student groups: one group uses the new testing framework, while the other group has no access to the detailed test reports. The results show a high acceptance of the new testing framework, which also is reflected in a distinct lower failure rate. Besides the positive feedback, the survey also indicates a shift from debugging code to solely applying tests.

Peer-review under responsibility of the organizing committee of HEAd´16

Keywords: Computer Science Education; Teaching Programming; Unit Tests.

1. Introduction

Students enrolled in introductory programming courses often draw on extremely different background knowledge from complete novices to profound programming experiences. Additionally dangerous smattering of knowledge can

* Corresponding author. Tel.: +49 89 6004 2612; fax: +49 89 6004 3412.
E-mail address: dieter.pawelczak @ unibw.de

Peer-review under responsibility of the organizing committee of HEAd´16
doi:10.1016/j.sbspro.2016.07.032

lead to misconceptions, which are hard to reveal. Despite the difficulties in teaching such a heterogeneous group, the number of students increase at our university in recent years resulting in larger groups in the lab. A measure against a rising failure/ dropout rate was the establishment of an automated grading system in the last four years. An important reason was the lacking capability of the instructors handling such a large group especially with regard to manually checking the source code. Another ratio is gaining more time for instructors to focus on programming issues than on the bureaucratic managing of checklists. An important aspect of the auto grading system was the integration into a single tool: an easy to use programming environment, that lowers the burdens for novice programmers; the same tool is used for live coding during the lecture and for submission of the programming assignments in the lab. Students can work on exercises at home or use the programming environment for their examination preparation. Besides these requirements, other features beyond the scope of this study are: visualization of data and control flow of C programs, availability for multiple platforms and an easy installation procedure. To fulfill these requirements we have developed the Virtual-C IDE[†].

In the first years of operating the auto grading system, the focus was to hide detailed information on testing from the students in order to avoid students to write code fitting the tests rather than coding according to the specification. Another advantage of this approach is, that students have to read the exercise description carefully to properly fulfill the requirements. On the other hand the system required a very good understanding from the course instructors to help students on reported failures. Due to the fact, that students often missed details in the exercise description or lacked a full comprehension of the required tasks, students often were discontent; especially in case an instructor could not sufficiently enough explain the failure of the test. To further enhance the system we introduced a JUnit like testing framework. In order to counteract the described problem of coding according to the test results, the number of tests and the test coverage are much higher compared to the previous system. Furthermore, random data is used to individualize test runs. The paper presents the results of a survey about the students' experiences with the auto grading system. Pawelczak, Baumann and Schmudde (2015) provide a detailed description of the testing framework integrated in the Virtual-C IDE.

2. Related Work

High failure/ and dropout rates are a common phenomenon for beginners' programming courses in engineering or computer science education. A lot of different measures and methods are discussed: from pushing better prerequisites – as DeLyser and Preston (2015) promote with early computer science education in high schools – to completely new programming course concepts or programming environments. Program visualization or visual programming environments are very interesting concepts and widely available for programming languages like Phyton or Java; compare for instance Berry and Kölling (2014). Block-C is a visual environment for the C programming language, as presented by Charalampos, Nektarios and Stavros (2015). Assembling programs visually by drag and drop prevents syntax errors and therefore lowers the barriers for novice programmers. As in our case the C programming course is a direct prerequisite for the embedded systems programming course, important educational objectives are the language syntax and the handling of a compiler environment. Another promising approach is incooperating tests in early programming courses. A survey of a testing first approach by Marrero and Settle (2005) did not indicate a major improvement in programming skills but shows, that by adding the testing concept students are forced to think more abstract. Janzen and Saiedian coined (2008) the phrase *test-driven learning* and found out, that testing first resulted in a better test coverage compared to a test last approach. Fidge, Hogan and Lister (2013) compared in an elaborate study the skills of writing tests to coding programs and were surprised, that students who produced good programs provided tests with a poor coverage. Edwards and Shams (2014) put up for discussion, that student programmers tend to write all the same tests. Bearing that in mind, we focus more on the advantages of using tests instead of writing tests. This is in consensus with Whalley and Philpott (2011) and their study on unit testing for novice programmers: Providing tests to students offers an additional feedback mechanism for students and supports completing assignments outside class time. We provide tests for auto grading programming assignments as well as

[†] https://sites.google.com/site/virtualcide/