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/

for exercises. For auto grading of lab work, students have no access to the test specification; still, the test report can directly be used to debug the program with the specific test data.

3. Data and methodology

3.1. Course Background

Data is collected for the first year C-programming course over the years 2014 and 2015. An introductory course on programming based on Java programming in NetBeans precedes this course. With varying concepts of the preceding course, student stated quite different prerequisites: in 2014 students were excellently prepared – the majority of students (64 %) stated in their own assessment a satisfactory programming knowledge. In the following we refer to these students as class *A*. In contrast in the following year about 69 % stated insufficient knowledge, here referred as class *B*. Other prerequisites were fairly equal, e.g. in both years about 10 % of the course participants were already skilled in C programming and about 30 % arrived with some knowledge on Java programming at university.

The programming course consists of 4 weekly hours lectures and 7 programming assignments in 4 groups of approximately 20 students. Each programming assignment has to be prepared at home with about 15 hours workload and additional 2 hours lab time. During lab time students submit their solutions in our auto grading system, which runs tests on their programs, provides feedback on errors and manages the overall student's achievements. The programming assignments included topics like integer and floating point operations, strings, arrays and pointers, structured data and dynamic data structures. For class *A* the system did not provide detailed test results, whereas for class *B* these were fully laid open.

3.2. Data Basis

The survey took place at the end of each of the courses. Out of about 60 questionnaires students returned 26 in 2014 and 32 in 2015 giving a total of 58.

4. Results

4.1. General Difficulties

In the first survey, we additionally asked about detailed difficulties with programming concepts, as e.g. listed in Lahtinen, Ala-Mutka and Järvinen (2005). Although our results correlate with the results in this publication, our students stated quite less difficulties. For instance, about 88 % state they had no or almost no problems with conditions and selection statements. These answers did not reflect the observation of the course instructors, as about every third student struggled in at least one of their assignments with malformed conditions or nested selection statements. About half of the students declared to have no or almost no problems with pointers in C; bearing in mind, that two assignments regarding string handling were related to pointers, these left a quite different impression for the instructors. So students claimed less difficulties as observed, which might be due to the fact, that the survey took place before the final examination. For this reason, we ask in the following year only for general difficulties, as shown in figure 1.

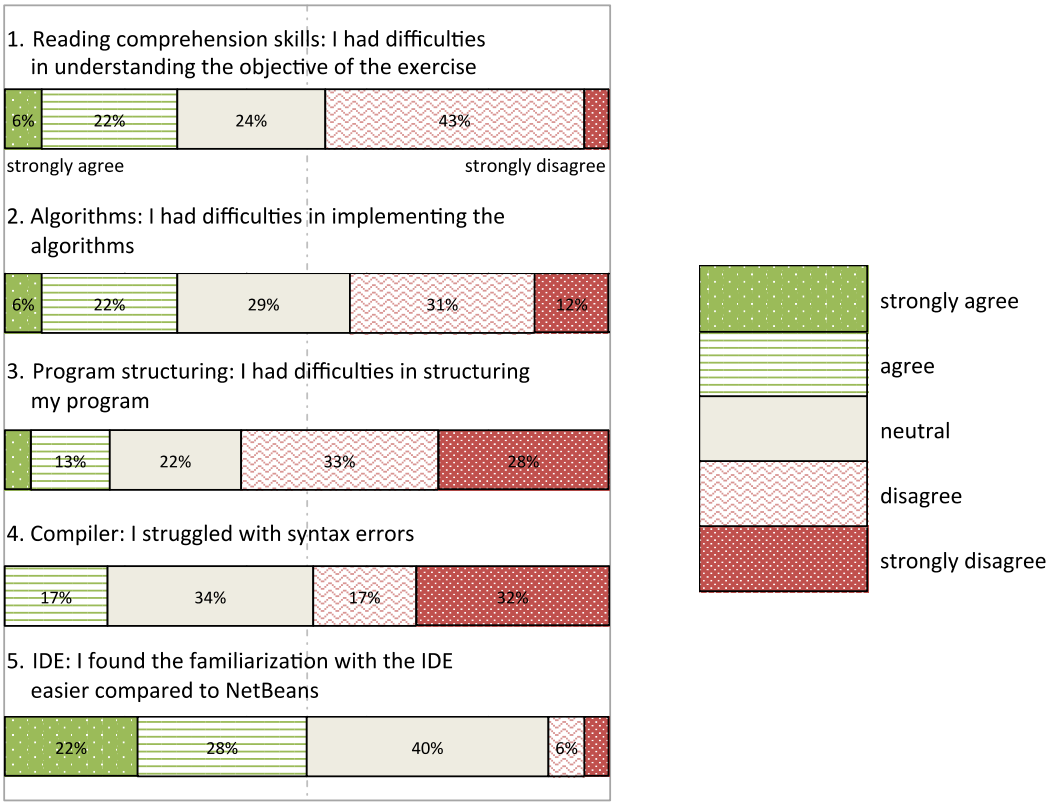

Fig. 1. Answers regarding the general environment and common difficulties during the course (Likert scale).

The first question is important as the programming assignments are checked automatically: thus a misunderstanding in the assignments tasks leads to a rejection of their submission. Only 30 % of the students claimed to had difficulties with understanding the objective of the exercise. This low number also explains the following two low percentages with respect to the program implementation (questions 2 & 3): the exercise description is quite stringent due to the formal testing methods and thus giving a quite clear instruction on the program's structure and the algorithms to use. Additionally the accompanying lecture discusses typical solutions to similar problems. A rather low number (17 %) struggled with syntax errors. About half of the students found starting to code with the Virtual-C IDE easier compared to NetBeans, a tool used in the preceding course. While NetBeans is a professional programming environment developed by Oracle Cooperation, the Virtual-C IDE in the lab is especially designed for programming beginners.

4.2. Materials and Support

As the preparation of the programming assignment mainly takes place at home, we asked for the different materials and approaches, that were helpful for completion of their programming assignments. While there is a broad agreement on the helpfulness of lecture notes and examples, the two classes differ very much in questions 8, 9 and 10: In class *A* 36 % strongly disagreed or disagreed, that discussions with fellow students were helpful, while in class *B* only 18 % gave that answer. Taking code snippets/ solutions from other students was for narrow 60 % of the students in class *B* helpful, while in class *A* only 29 % agreed or strongly agreed, that these were helpful. On average still 50 % rather agreed in this matter. It might be due to the composition of the classes, but it might also indicate, that less prepared students tend to rather copy code than to write it on their own, compare e.g. Cosma, Joy and

Sinclair (2011). The last question is especially important, as debugging is an important task taught in the accompanying lectures. While the preceding course of class *A* extensively discussed debugging, students from class *B* had about no experiences with debugging. Thus class *A* claimed, that debugging helped to 72 % (agree & strongly agree) and 12 % stated, that it rather did not help. 22 % of the students from the less experienced class *B* found that debugging rather did not help and only 50 % agreed or strongly agreed on debugging being helpful.

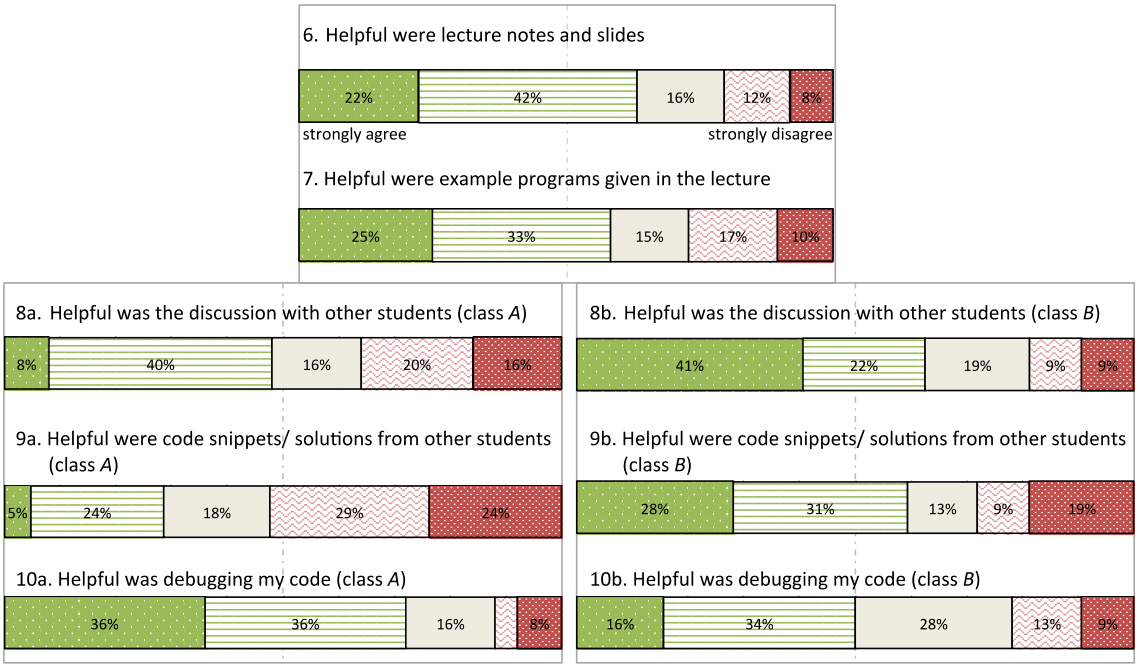

Fig. *2.* Survey results with respect to the resources/ media used for solving the programming assignments.

4.3. Debugging and Testing

About 76 % claimed, that debugging is good for understanding the control flow. This properly reflects the concept of live-debugging in the lectures, compare figure 3. Still as discussed in the section 4.2, considerably less students agreed, that debugging was helpful. Many students gave an neutral answer when comparing the debugging between the IDE and NetBeans, which is quite obviously, as in both environments the concept of debugging is very similar. Again, there are quite differences in both classes: class *B* stated with 60 % of the students, that the debugger of the IDE is easier to handle, which is due to the fact, that these students were less experienced with debugging.

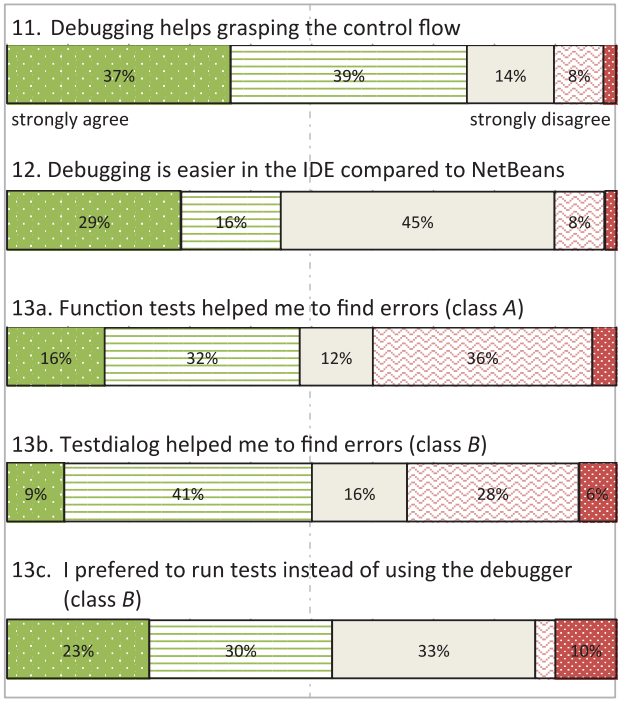

Fig. 3. Survey results regarding debugging and the testing framework.

The system based on the new testing framework is highly accepted: 83 % agreed or strongly agreed to enjoy working with the system with the integrated test dialog, compare figure 4, question 14b (class *B*). The year before only 40 % of the students liked the electronic submission system without the testing framework. Although our impression was already during the course of class *A*, that competing against the machine triggered students motivation, students felt more comfortable with detailed feedback and concise hints on the failures. Interestingly, more students from class *A* (58 %) could imagine to perform the course online from home without the help from instructors, compared to 45 % in class *B*. Class *B* even voted with 42 % against such a different lab design. Several indicators might lead to this constellation. Certainly an effect had a highly dedicated instructor of class *B*, that didn't taught in class *A*. Other reasons are certainly found in the different prerequisites of the classes and the fact, that testing as like debugging requires additional skills to standard coding.

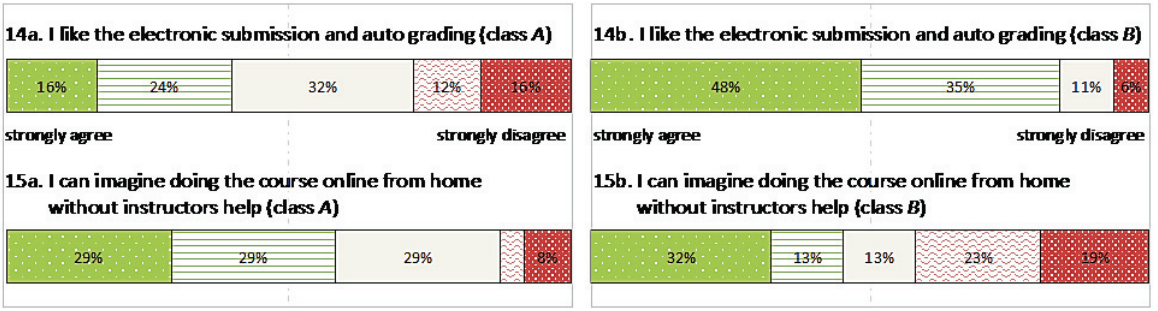

Fig. 4. Acceptance and further development of the auto grading system.

5. Conclusion and Outlook

The incooperation of our new testing framework had a positive effect on the failure rate in the final examination of our programming course: students of class *B* completed their exams with a failure rate of less than 18 % (even with less prerequisites) compared to about 29 % for students without the support of the testing framework. Although there are many different influences on the failure rate, we found an important aspect in the student's motivation: the transparent disclosure of tests rather encourages student's comprehension of their failure and students adapted more willingly their code. As many tests refer to typical programming patterns (e.g. comparing a parameter with NULL), students were more skilled on common code snippets in the examination. We also experienced, that running tests reduced the need to debug the programs. Superior students often realized from the test results, what mistake they made and what code portion they need to change. On the other hand, less experienced students fiddled with their programs by repeatedly running tests without debugging or reflecting the program's behavior. Only a few students used the possibility to debug their program by applying the test data as specified in the test report. We assume, that this is due to the fact, that the second group (class *B*) had less prerequisites with regard to debugging. It might also indicate a lack, that students do not grasp the overall picture: how useful debugging and testing can be brought together.

Our future research is focused on testing the code quality, as most tests succeed, even if the code is cumbersome. Although the system supports performance tests, the implementation of such tests is quite elaborate. We examine a structured dynamic analysis on the code under test during the test run.

References

Berry, M., & Kölling, M. (2014). The state of play: a notional machine for learning programming. In *Proceedings of the 2014 conference on Innovation & technology in computer science education* (ITiCSE '14). ACM, New York, NY, USA, 21-26. doi:10.1145/2591708.2591721

Charalampos, K., Nektarios, N., & Stavros, C. (2015). Block-C: A Block-Based Visual Environment for Supporting the Teaching of C Programming Language to Novices. In *Proceedings of the 9th International Conference on New Horizons in Industry, Business and Education (NHIBE 2015), Skiathos Island*, Greece, August 27-29, 2015, 160-166

Cosma, G., Joy, M., & Sinclair, J. (2011). Source Code Plagiarism—A Student Perspective. IEEE Transactions on Education, 54 (1), 125-132. doi: 10.1109/TE.2010.2046664

DeLyser, L. A., & Preston, M. (2015). A Public School Model of CS Education. In *Proceedings of the Intern. Conf. on Frontiers in Education: CS and CE (FECS'15)*, Las Vegas, Nevada, USA, July 27-30, 2015, 233-238.

Edwards, S. H., & Shams, Z. (2014). Do student programmers all tend to write the same software tests? In *Proceedings of the 2014 conference on Innovation & technology in computer science education (ITiCSE '14)*. ACM, New York, NY, USA, 171-176. doi:10.1145/2591708.2591757

Fidge, C., Hogan, J., & Lister, R. (2013). What vs. how: comparing students' testing and coding skills. In *Proceedings of the Fifteenth Australasian Computing Education Conference - Volume 136* (ACE '13), Angela Carbone and Jacqueline Whalley (Eds.), Vol. 136. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 97-106.

Janzen, D., & Saiedian, H. (2008). Test-driven learning in early programming courses. In *Proceedings of the 39th SIGCSE technical symposium on Computer science education* (SIGCSE '08). ACM, New York, NY, USA, 532-536. doi:10.1145/1352135.1352315

Lahtinen, E., & Ala-Mutka, K., & Järvinen, H.-M. (2005). A study of the difficulties of novice programmers. SIGCSE Bull. 37, 3 (June 2005), 14-18. doi:10.1145/1151954.1067453.

Marrero, W., & Settle, A. (2005): Testing First: Emphasizing Testing in Early Programming Courses. ACM SIGCSE Bulletin, 37(3): 4-8. doi:10.1145/ 1151954.1067451

Pawelczak, D. & Baumann, A., & Schmudde, D. (2015): A new Testing Framework for C-Programming Exercises and Online-Assessments. In *Proceedings of the Intern. Conf. on Frontiers in Education: CS and CE (FECS'15)*, Las Vegas, Nevada, USA, July 27-30, 2015, 279-285

Whalley, J., & Philpott, A (2011). A unit testing approach to building novice programmers' skills and confidence. In *Proceedings of the Thirteenth Australasian Computing Education Conference* (ACE 2011), Perth, Australia, 113-118