Innovative Approaches in C Introductory Programming Courses

Matej Madeja[1][0000−0002−8197−1962] and Jaroslav Porubän[2]

Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Letná 9, 042 00 Košice, Slovakia
matej.madeja@tuke.sk[1], jaroslav.poruban@tuke.sk[2]

Abstract. The choice of programming language in an introductory programming course (CS0) is crucial for understanding of basic programming principles. From our previous research, C programming language seems to be the most appropriate for these courses. In this paper we propose 8 problem sets (PS), the one of whom focused on Arduino boards, and we implement them to the curriculum of an CS0 course in C which is yearly attended by 500 students. We have collected PS results from four semesters of the course and we observed the impact of PS quantity on students' motivation and overall results. From our observation students mostly start to work since the last two weeks before the deadline, regardless of the solution evaluation length. A surprising result was the fact that with a higher number of PSs in a semester the average rate of submissions and final results was lowered. At the same time, we confirm the positive impact of game programming in CS0 courses that motivate students to work from the early beginning of testing and it is 27% earlier than in the case of classical PSs.

Keywords: computer science courses · CS0 · programming · C language · problem set · innovative approaches.

1 Introduction

Computer science and software engineering have become very popular at universities around the world due to the constant lack of IT engineers in the market. Universities respond to this by expanding demanded study fields, and thanks to that massive open online courses (MOOC) has grown considerably [1]. This boom causes IT departments can be attended by students who are not motivated enough to study such discipline and it is the role of universities to make their courses more attractive. Introductory programming courses, often labeled as CS0 or CS1, have an important role in motivating students for future work in software engineering.

Universities exploit mentioned situation and try to attract students to less popular study fields through IT departments, so they often accept all students without admission procedure. In this case, it is necessary to classify the most prospective students for considered study field. If the course is attended by a student who has earlier programming experience, he often underestimate the role of CS0 courses relying on his previous experience. This student brings his own writing code habits and thinking, which, according to our experience with teaching, are often negative for such students. Many authors in their papers [2,3] confirm that the programming style that students learn as first influences their thinking and creation of the code for a long time. The aim of these courses is therefore to broke bad and learn the engineered correct approaches for program architecture and code habits, such as naming conventions, algorithmization and creation of comprehensible source code.

As we can see, the task of introductory courses is very crucial for the future development of the student. According to Allain [4], author of a book of the C family, there are 5 rules leading to the correct learning of the basics of programming: 1. study enough examples, 2. run code and test real examples, 3. write own source code as often as possible, 4. use a debugger and 5. constantly look for new sources for study. Given the above recommendations, we would like to modify the current CS0 course in C language at our university, suggest changes in curriculum through problem sets (PS) and continuously monitor students' performance and results in other programming courses during their studies. At the same time, we try to motivate students to work continuously during the course. Our main hypothesis for this research is:

Hypothesis 1: *A large number of difficult assignments in introductory programming courses have a positive impact on student results at other university programming courses.*

In this paper we provide some partial results for the future hypothesis verification we formulated. Since we need basic information about students' motivation in CS0 course to compare with other courses at the university, we formulate the following research questions focused on the CS0 course *Programming*:

RQ1: Do students work more continuously if they have more time to develop a solution for particular problem set?
RQ2: Does the assignment quantity affect the number of student submissions rate and final results?
RQ3: Are students more motivated when programming games or motivation is similar to classical problem sets?

The following sections we describe current recommendations and research in this area as well as the design and partial results of the performed experiment.

2 Related Work

For a research in university courses, one must first look at the pedagogical point of view, therefore in what form and in what structure information should be provided to the student. That is the reason we divided this section into pedagogical methods and experiments in university courses.

2.1 Pedagogical Methods

Pecinovský in his book *OOP - Learn Object Oriented Thinking & Programming* [5] and in his article [6] defines the *Architecture First* methodology, in which he advises to teach by the top-down approach, from the overall program architecture to code details. His method is based on the Berin's Early Bird approach [7]. Pecinovský, in his teaching programming research, also recommends to start programming real programs as soon as possible, making PS attractive as possible, example using games. In our approach, we build curriculum on the foundations of Pecinovský with respect of Bergin, trying to guide students from general programming issues to a detailed understanding of programming principles using real applications or rather problem sets.

Pair programming is a technique where two students collaborate on the same project. In [8] authors confirm the positive impact of this approach on retention, confidence and program quality. Despite the positives, we consider this approach to be inadequate for CS0 courses because this way not all students will meet the same issues.

2.2 Experiments

Hickey [9] has implemented a multi-year experiment in which he obsereved the use of schema-based web programming in a CS0 course. However, in his approach combines teaching in a specific language and a broad overview of computer science, which includes general topics such as hardware, software and history. This paper also summarizes multi-year students' results, but we focus directly on programming in the particular language (C) and exclusively for students of computer science or software engineering, thus our results are directly pointed on students of informatics

Use of *Games First* approach was addressed by Leutenegger and Edgington [10], who show the positive impact of games on motivating students in introductory courses. At the same time, the author appreciates the degree of understanding of the basic concepts by students. Their approach was an inspiration for our designing new PS and curriculum design. Like many other experiments, this also focuses on relationships between gender and success in the course. On the contrary, we look at student results over several years, regardless of gender, and examine whether the type of assignment or particular task affects student motivation and result.

Alvado et. al. [11] examined the relationship between student experience and their success in the pair programming course. They found that past experience mostly affects the rate of success positively, to a greater extent in men than in women.

Matzko and Davis [3] used an approach of learning course concepts using real-world image processing problems in C. Their results of the questionnaire claim that this approach was excellent for students. From our experience, the student's subjective opinion on the course (colleced by the questionnaire) often does not correspond to his or her real benefit to the student, on the other hand, we think their results represent student motivation in the course.

3 Original course status

In 2014, focusing on the problem sets, we noticed the obsolescence of a CS0 course at our university. Starting in 2013, we started using automated tests to evaluate assignments in the course. Despite our attempt to motivate students to better and more successful results by having multiple evaluations of their assignment, we have observed the same problem as during the manual evaluation - students submitted their solutions at the last minute. At the same time, only 2 assignments were included in the course, so the course offered students more theoretical and less practical experience.

Another problem was the poor connection between the lecture, the labs and the problem sets. The students had often necessary knowledge to complete the assignment since the last week before the deadline, so they were unable to work continuously. At the same time, the disadvantage was that the labs were mainly used to consult the students' solutions, therefore, the labs potential was not sufficiently exploited.

4 Designing a New Course Organisation

Based on the problems of the original course curriculum and according to the recommendations mentioned in section 2, we decided to change the CS0 curriculum at our university in 2015. During making big changes in the course we also considered the suitability of C language for CS0 courses.

4.1 Programming Language Selection

Although the *Programming* course was taught in the C language from the beginning of its existance, we registered the increased interest of other universities in other languages, so we were also thinking about changing the programming language. We therefore conducted an analysis [12] in which we compared world and local universities so that we could compare the suitability of each language. At the same time, we have studied the impact of the choice of language on problem sets of the course and how it can influence the understanding of basic contexts in programming.

We found out that 70% of 37 top universities in the USA use Python in CS0 courses. This has gradually changed over the last decade and we consider it as a result of simpler understanding. With increasing popularity of object-oriented programming (OOP), student's comprehension is higher thanks to simpler real-world objects mapping to the source code. We can say that increasing abstraction simplifies program comprehension. The choice of language for universities is likely affected by trying to provide programming in a language that is very popular and perspective for the student's future.

On the other hand, we see a big gap in understanding how low-level programs work. The low-level language, such as C, which is the 2nd most popular in CS0 courses, includes topics such as structures, pointers, static and dynamic memory, stack overflow, etc., which have a very positive impact on the understanding of programming contexts. Based on our previous analysis in [12] and the mentioned benefits, we consider C to be very suitable for CS0 courses for software engineers.

4.2 Proposal of problem sets

Inspired by different universities in the world or interesting programming competitions (e.g., ACM ICPC[1])), we have decided to split the problem sets into 2 categories: games and classical problems to solve. We wanted to evenly spread problem sets in the course for these two categories in order to observe the impact of the type of assignment and its number in the course on the submission rate and the final score of the student.

The goal of the new PSs is a better distribution in the course, so that students are motivated to work continuously during the semester. The design of several smaller tasks within the semester indirectly influenced the final curriculum of the course, so each problem set could be adapted to the lecture content. We also designed moderated scenarios for the labs to improve students's cooperation with the teacher by solving an issue together.

For the 4 semesters of the course in the period 2015-2018t following PSs were proposed:

K An 2048 game alternative. We use the character version of the game so students can not use existing solutions. The student must create the whole game, including screen rendering and hall of fame. Goal: arrays, structures, game loop.

Top Secret Classical ciphers assignment. The task is to encrypt or decrypt a string using a defined sequence of simple ciphers, such as playfair and our own ciphers (using bit masks, chain variations). Goal: arrays, strings, bit operations, pointers.

Adventure A text adventure game or interactive fiction game. The task of the student is to create the whole game, i.e. rooms, items, player and other extentions. Goal: bigger project maintenance, unions, dynamic memory, bit masks, regular expressions.

Tiktak & Triangulation Tiktak is a simple game (not considered as game in this experiment) with listing of numbers count requested by the user. If the actual printed number is divisible by 3 is typed `"tik"`, if by 2 `"tak"`, if both `"tik tak"`. In triangulation the student has to calculate the correct coordinates of the enemy ship against the cannon on the land. Goal: working with expressions, using standard libraries.

Tesco Simulation of a store self-service cash register. The student has to deal with the emerging inaccuracies of type `double`, for example when issuing coins. Goal: representation of numbers, return values.

Hangman A game of guessing a random word by player. If the word is not guessed into a defined number of attempts the game ends. Goal: lists, arrays, game loop.

[1] https://icpc.baylor.edu/

Problems to Solve It contains 4 complex algorithmic tasks. Goal: algorithmization, stdin, stdout.

For all PSs also test suites have been created to evaluate students' solutions. In 2017 and 2018, we used *Arduino board* as one of the assignments, where student had to implement the game *Mastermind*. In this way, students are in touch with the electronics programming, which can increase their motivation for programming Internet of Thinks (IoT) in the future. The results of PS *Mastermind* are not included in this paper, as we focus exclusively on the results from the automated tests. The full text definitions of all PSs can be found at the official course website[2] in slovak.

5 Method

With the implementation of new problem sets from section 4.2 into the course, a testing environment called *Arena* has been developed. This environment automatically downloads student solutions from the Gitlab version control system (VCS) and tests them according to the defined structure. To assess the impact of the type of assignment and the number of assignments within the semester on the students' motivation and results, we use the biggest course at our university (approximately 500 students/semester), so it provides a diverse sample for us.

From 2015 we have 4 semester results of total 2090 students. For each year, we selected different PSs combination and always with similar game/classic problem set ratio. At the same time, we have distributed different types of PSs during semester evenly so the results tracking at the beginning/end of the semester has no negative effect on the results of particular PS type. As usual, at the end of the semester students are working on assignments from different courses at once, so if all the PS games are at the end of the semester, it could negatively affect the results.

We have always adapted the number of assignments to the speed and focus of lectures. This way we tried to preserve following presentation of the curriculum: 1. lecture, 2. labs, 3. problem set. We also tried to sort PSs in a way to create better interconnections of new knowledge. Use of proposed PSs in individual years and their order during the semester can be seen in the table 1.

As can be seen, some PSs were used only in 2015. These assignments were less complex, so they were moved to another course and in later years we used more complex tasks.

6 Partial Results

In total, we collected 64,825 submissions of students. For PS, which was repeated in different years, there was also a different testing period for the solution. Students could submit the solution for evaluation at any time, but the tests were run every 3 hours and only if the student made a change to the solution. General statistics of all years are shown in Table 2.

[2] https://kurzy.kpi.fei.tuke.sk/pvjc/

Table 1. Problem set order and use in previous course years.

Problem Set	**Type**	**Order of PS in year**			
		2015	**2016**	**2017**	**2018**
K	game	4	1	1	1
Top Secret	classic	5	2	2	2
Adventure	game		3		4
Tiktak & Triangulation	classic	1			
Tesco	classic	2			
Hangman	game	3			
Problems to Solve	classic				3

Table 2. General stats about course and problem sets assessment in particular years.

Year	**Number of students**	**Avg. score (100 = full)**	**ATiD*to reach max. score before deadline**	**ATiD*of assignment submission posibility**
2015	543	56.80	8.24	18.40
2016	325	63.56	9.02	18.67
2017	637	60.65	4.61	9.50
2018	585	51.62	4.91	12.50

* ATiD = Avg. time in days;

To answer the **RQ1** we have tracked the submission rate of solutions over time. In the image 1 we can see that even with a larger time for submission students start working since the last 2 weeks before the deadline and rarely students have used the possibility of an earlier submission. An interesting finding is that the highest peak of submissions is 1 day before deadline, while in general the last week is the most productive for students, both of submission rate and the increase of final score.

Result 1: Student's ongoing work is not affected by the amount of time to submit the solution, students work mostly last 2 weeks before the deadline.

To compare the submission rate and results (**RQ2**) in different years we calculated for each problem set the average submission rate per year/PS/student (figure 2). As we can see, the submission rate in years with a higher number of PSs is lowered, but this result is ambiguous. When analyzing the submissions of particular PSs individually, we found the negative impact of using the PSs from the previous semesters of the course on these results. Despite the changes to the definition of the assignment, the number of submissions has been reduced in most cases for unchanged tasks, probably due to the plagiarism of solutions from previous years. When observing the tasks used for the first time the students worked more actively, because the student was not influenced by the correct solution (from the previous year's course). With a higher number of PS in the semester the average submission rate decreases and the overall results in score

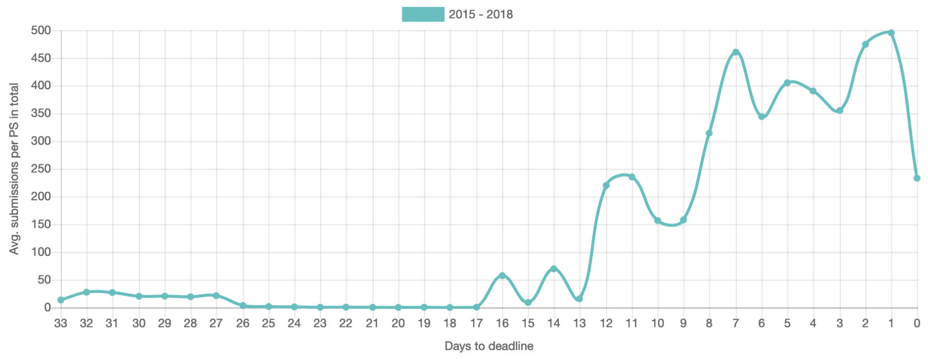

Fig. 1. Average number of submissions per PS for a particular day before deadline in the last 4 runs of the course.

are worse. However, the worse results do not represent the student's learning experience, because with a greater number of issues the student meets, it can help him understand the context better.

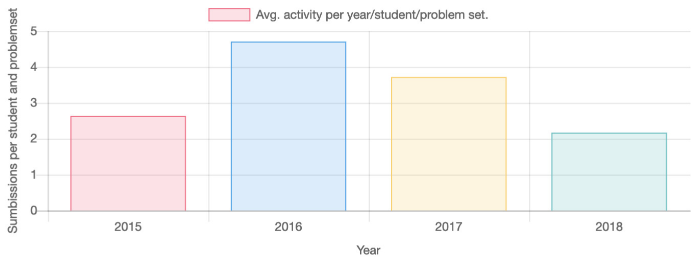

Fig. 2. Comparison of the average submission rate per year, problem set and student.

Result 2: The more problem sets a student has to develop in the course, the average submission rate is lowered and the final score is worsened.

A good indicator of ongoing work of students we consider the number of submission per day. If the student is sufficiently motivated it is assumed that he or her works on the solution earlier. We found out that students are more active when programming games (**RQ3**) rather than programming classical PSs (see figure 3). When we analyzed the results of individual PSs, games were the fastest growing in the beginning of the chart, indicating an increased student motivation. Subsequently, the relatively similar number ($\pm$ 500 submissions/day) of attempts to evaluate the assignment was counted. When observing the graphs of individual classical PS we saw a gradual increase in interest and the peak was reached just before the deadline, which means low student motivation. The average score on game programming was on average 3.19% higher over the overall evaluation of the award, but we consider this difference as not significant.

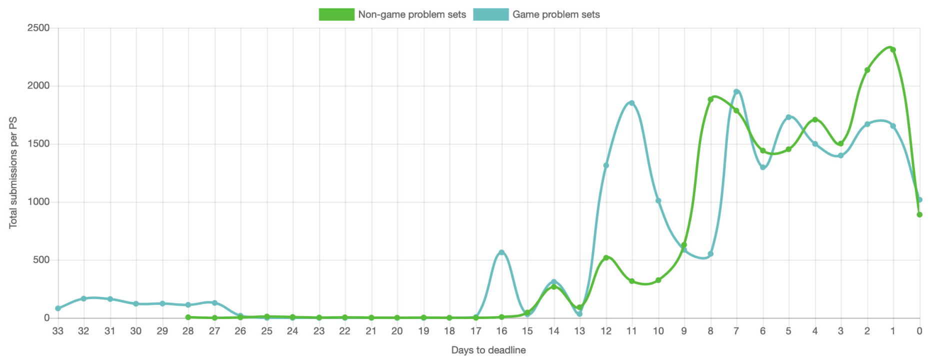

Fig. 3. Comparison of submission frequency by days to deadline for game vs. classical problem sets.

Result 3: Students are more motivated developing games and they work from the beginning of the PS publication. While programming a classical PS is the motivation initially lower, reaching a similar level in 27% later of total evaluation time.

7 Conclusion and Future Work

Introductory programming courses play an important role in the way of how students think and develop the source code, both from a pedagogical point of view and previous empirical studies. The actual impact on students for following programming courses after CS0 has not been quantified in such way. In this paper, we answer partial questions about students' motivation and continuously work in the introductory programming courses, with 4-year observation of students in the biggest course at our university. From our observation, students start to work since the last 2 weeks before the deadline, regardless of the length of the solution evaluation. For authors, the surprising result was the fact that with a higher number of PSs in the semester the average rate of submissions and final results was lower. At the same time, we confirm the positive impact of game programming in CS0 courses that motivate students to work from the early beginning of testing and it is 27% earlier than in the case of classical PSs.

In our future work, we would like to include additional assignments in the course in order to achieve more relevant results, as we noticed the negative impact of PS reusing. We also would like to take into account the results of the Arduino PS and evaluate the impact of CS0 courses on the results in other programming courses. Thanks to students' solutions availability at VCS, we can take a closer look at programming styles and habits, to improve students' source code comprehension.

Acknowledgment

The authors would like to thank all the participants in the research, especially students and teachers of course *Programming* of *Technical university of Košice*. This work was supported by project KEGA No. 053TUKE-4/2019: Learning Software Engineering via Continues Challenges and Competitions.

References

1. Cook, M.: The 50 most popular moocs of all time (04 2015) http://www.onlinecoursereport.com/the-50-most-popular-moocs-of-all-time/.
2. Robins, A., Rountree, J., Rountree, N.: Learning and teaching programming: A review and discussion. Computer science education **13**(2) (2003) 137–172
3. Liberman, N., Beeri, C., Ben-David Kolikant, Y.: Difficulties in learning inheritance and polymorphism. ACM Transactions on Computing Education (TOCE) **11**(1) (2011) 4
4. Allain, A.: 5 ways you can learn programming faster (2011) https://www.cprogramming.com/how_to_learn_to_program.html.
5. Pecinovský, R.: OOP - Learn Object Oriented Thinking & Programming. Academic series. Tomas Bruckner (2013)
6. Pecinovský, R.: Methodology architecture first. **5** (04 2013) 107–114
7. Bergin, J.: Fourteen pedagogical patterns. In: EuroPLoP, Citeseer (2000) 1–49
8. McDowell, C., Werner, L., Bullock, H.E., Fernald, J.: Pair programming improves student retention, confidence, and program quality. Commun. ACM **49**(8) (August 2006) 90–95
9. Hickey, T.J.: Scheme-based web programming as a basis for a cs0 curriculum. SIGCSE Bull. **36**(1) (March 2004) 353–357
10. Leutenegger, S., Edgington, J.: A games first approach to teaching introductory programming. In: Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education. SIGCSE '07, New York, NY, USA, ACM (2007) 115–118
11. Alvarado, C., Lee, C.B., Gillespie, G.: New cs1 pedagogies and curriculum, the same success factors? In: Proceedings of the 45th ACM Technical Symposium on Computer Science Education. SIGCSE '14, New York, NY, USA, ACM (2014) 379–384
12. Madeja, M.: Innovative approaches in introductory programming courses. Master's thesis, Technical university of Košice (05 2015)