Teaching Computing in primary school: Create or Fix?

Tom Neutens
IDLab, Department of Information Technology, Ghent University - imec
Ghent, Oost-Vlaanderen, Belgium
Tom.Neutens@UGent.be

Francis wyffels
IDLab, Department of Information Technology, Ghent University - imec
Ghent, Oost-Vlaanderen, Belgium
Francis.wyffels@UGent.be

ABSTRACT

We describe the setup of an experiment conducted in the spring of 2018. The experiment aims to identify the differences between teaching computing through creating programs as opposed to fixing them. Quantitative and qualitative data are collected in the forms of pre- and post- knowledge tests, pre- and post- attitude tests, programming environment logging data and student interaction video's. The logging data shows that starting from incorrect program leads to less interactions with the code. However it leads to more testing in simulation and on the physical system.

KEYWORDS

Computer science education, STEM education, Creating, Debugging, physical computing

ACM Reference Format:
Tom Neutens and Francis wyffels. 2018. Teaching Computing in primary school: Create or Fix?. In *Proceedings of the 13th Workshop in Primary and Secondary Computing Education (WiPSCE '18), October 4–6, 2018, Potsdam, Germany*. ACM, New York, NY, USA, Article 4, 2 pages.

1 INTRODUCTION

Improving our knowledge about primary computer science (CS) education is key to enabling its implementation in the classroom. The way teachers decide to teach a specific CS topic in the classroom has consequences in multiple dimensions. These dimensions include: learning efficiency, motivation, attention, level of understanding, time required, course materials and many others. Identifying which educational design strategies work in certain situations and which do not is important to provide teachers with sufficient theoretical knowledge when choosing an implementation.

Several studies have aimed at identifying better methods for teaching CS. In [5] the authors examine the effect of subgoal labels in a K-3 LightBot course. They showed that subgoal labels significantly improve the speed at which students can progress through exercises. The authors of [1] show the importance of teaching a conceptual framework of program execution to improve performance in the KODU programming exercises. The learners who were introduced to the framework were more capable at explaining and predicting a program its behavior. In [3] the authors analyze which CS concepts should be thought at what age.

Postdam, October 2018, Potsdam, Germany

In the CS education community it is generally believed that teaching computing does not only provide students with knowledge about computing but also trains them in higher level concepts. These concepts are mostly grouped under the umbrella term *computational thinking*. Many definitions of computational thinking exist in literature, one of the more popular definitions is the one by J. Wing [8]. She defines computational thinking as: *The thought processes involved in formulating a problem and expressing its solution(s) in such a way that a computer -human or machine- can effectively carry out.* In practice computational thinking is split into multiple dimensions. These dimensions are not consistent in literature however, most include: abstraction, decomposition, evaluation, algorithms and logic. Training students in mastering these computational thinking skills is important since it should enable them to apply these principles in many different contexts. Consequently, it is essential to identify which teaching techniques affect which computational thinking principles.

Previous research has shown that learning debugging skills, apposed to just learning how to program, are transferable to other domains [2]. However, debugging requires a deeper understanding than just writing a program from scratch. This might detrimental to the motivation of students who learn about programming through debugging [6]. Consequently, we designed an experiment to assess the effect of two different teaching methods on the learners' computational thinking ability and attitude. One teaching method focuses on learning to program by creating applications from scratch. The second method teaches them the same concepts but does this by letting the students debug the code provided to them.

2 EXPERIMENTAL SETUP

To conduct our experiment we designed two similar robotics workshops. These workshops both consist of three two hour sessions. The content and structure of both workshops is exactly the same. However, in the first workshop students write their code from scratch, the second workshop provides the students with an incorrect program and challenges them to fix the code. Both types of the workshop were given to five different groups. These groups have a size varying between 14 an 24 primary school students aged 10 to 12. In total 203 students participated in our experiment, 105 students participated in the "create" workshop the other 97 students in the "fix" workshop. The workshops were conducted in different schools across the country by two teachers. One of the teachers always taught the first session of the workshop the other teacher the second and third session. The three sessions of each workshop were conducted within the same week.

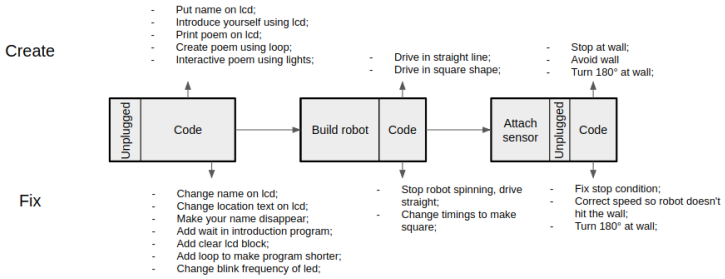

Figure 1: Overview of the create and fix workshops. The workshop structure is similar, only the coding exercises are different. Inside the blocks the session structure is shown. Above and below the coding exercises for each session type are shown.

2.1 Physical computing workshop

The workshop was designed in a physical computing context. Physical computing has the advantage that it can be easily linked to STEM education which has recently been added to the regional curriculum. Additionally, physical systems have been shown to have a positive effect on motivation because it supports active learning [4]. Within this context we designed two workshops, one in which the students write all code from scratch and one where they get an incorrect solution to the problem which they have to fix. We will refer to the different workshops as the create- and fix-workshops respectively. We designed both the create- and fix-workshop using a similar structure show in Figure 1.

2.2 Data collection

To measure the effects of the different workshops we collected many different types of data. (1) All participants take a pre- and post- knowledge test. This test consists of five Bebras questions and four basic programming questions. The programming questions assess their understanding of the learned programming concepts and the Bebras questions measure the effect on general computational thinking knowledge. (2) During five of the ten sessions (three create and two fix) a thesis student joined the workshop and filmed the interactions of two participating groups. This data will be used to perform a qualitative analysis of the workshops. (3) The students took a pre- and post- attitude test. The test measures the attitude towards engineering. This test is based on the work of Summers et al. on validation of attitudes towards science [7]. (4) To learn more about how the students interact with the programming environment and determine the strategies used to solve the different problems, all interactions with the programming environment are logged. All button clicks as well as code edits are timestamped and saved to a database.

3 LOGGING RESULTS

The collected dataset contains 567408 logging entries, 363793 from the create group and 203615 from the fix group. In table 1 an overview of the number of times certain events occurred in each workshop type. The table shows that there was a significant difference in the interactions with the environment between the create and fix group. The create group had to build the code from scratch so logically it had more code edits. When we compare the number of runs inside the simulator, we see a significant difference since the create group was not instructed to use it. However, some students found their way to the simulator most likely by accident. Consequently, the number of physical runs is higher in the create group. However. If we add the number of simulated and real runs we can see the fix group had more runs (5682) than the create group (4948).

Event	Create	Fix
Code change	56563	33834
Start simulation	60	3430
Program physical robot	4888	2252

Table 1: The number of log entries for selected events.

4 CONCLUSION

This abstract describes the setup of an experiment executed in the spring of 2018. We have provided some preliminary statistics extracted from our logging data. These show that starting from incorrect program leads to less interactions with the code. However it leads to more testing in simulation and on the physical system. A more extensive analysis of all data will be available in later publications.

REFERENCES

[1] Ashish Aggarwal, David S Touretzky, and Christina Gardner-McCune. 2018. Demonstrating the Ability of Elementary School Students to Reason about Programs. (2018).

[2] Sharon McCoy Carver. 1986. Transfer of LOGO Debugging Skill: Analysis, Instruction, and Assessment. (1986).

[3] Diana Franklin et al. 2017. Using Upper-Elementary Student Performance to Understand Conceptual Sequencing in a Blocks-based Curriculum. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM, 231–236.

[4] Scott Freeman et al. 2014. Active learning increases student performance in science, engineering, and mathematics. *Proceedings of the National Academy of Sciences* 111, 23 (2014), 8410–8415.

[5] Johanna Joentausta and Arto Hellas. 2018. Subgoal Labeled Worked Examples in K-3 Education. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. ACM, 616–621.

[6] Zhongxiu Liu et al. 2017. Understanding problem solving behavior of 6–8 graders in a debugging game. *Computer Science Education* 27, 1 (2017), 1–29.

[7] Ryan Summers and Fouad Abd-El-Khalick. 2018. Development and validation of an instrument to assess student attitudes toward science across grades 5 through 10. *Journal of Research in Science Teaching* 55, 2 (2018), 172–205.

[8] Jeannette M Wing. 2008. Computational thinking and thinking about computing. *Philosophical transactions of the royal society of London A: mathematical, physical and engineering sciences* 366, 1881 (2008), 3717–3725.