Quantifying Patterns and Programming Strategies in Block-based Programming Environments

Max Kesselbacher
Department of Informatics Didactics
University of Klagenfurt
Klagenfurt, Austria
max.kesselbacher@aau.at

Andreas Bollin
Department of Informatics Didactics
University of Klagenfurt
Klagenfurt, Austria
andreas.bollin@aau.at

***Abstract*—Pupils are often first exposed to programming in block-based programming environments like Scratch. Identifying and measuring the previous experience of students learning to program is a key to improve the teaching of programming. In this contribution, we outline an approach to measure and evaluate programming interactions with the block-based programming environment Scratch. First results, obtained with eight upper secondary school students, show that programming skills and patterns can be quantified with interaction metrics measured during program construction. The aim is a more fine-grained identification and assessment of programming skills.**

***Index Terms*—block-based programming, programming skills, programming patterns, learning analytics**

I. Introduction & Motivation

Is learning to program hard, or is it easy? In a recent paper, Luxton-Reilly [1] argues that the answer has many facets. Children have been quite successful in programming computers for decades [2], using different languages and environments. But programming is still perceived as a difficult skill to learn, and at university level the phrases *failure rate* and *introductory course* are almost synonymous [3]. Luxton-Reilly concludes that the learners' abilities have to be considered.

With the advent of computational thinking [4], many national curricula for primary and secondary schools included programming as a standard part of education. In most cases, pupils and students are first exposed to visual and block-based environments like Scratch [5] as they demonstrated to be well accepted at their developmental stage. However, such languages differ in the expressiveness of text-based languages, might limit the future developers' view on computer programming at higher education, and can influence the acquisition of programming skills. To improve teaching, a closer look at the previous knowledge of freshman programmers is needed. Learning to program in Scratch has been attributed to *positive* (fewer learning difficulties [6]) and *negative* (code smells that hamper learning [7]) effects regarding the transition to text-based programming. Meerbaum-Salant et al. [8] identify, but do not quantify, undesirable programming habits in Scratch.

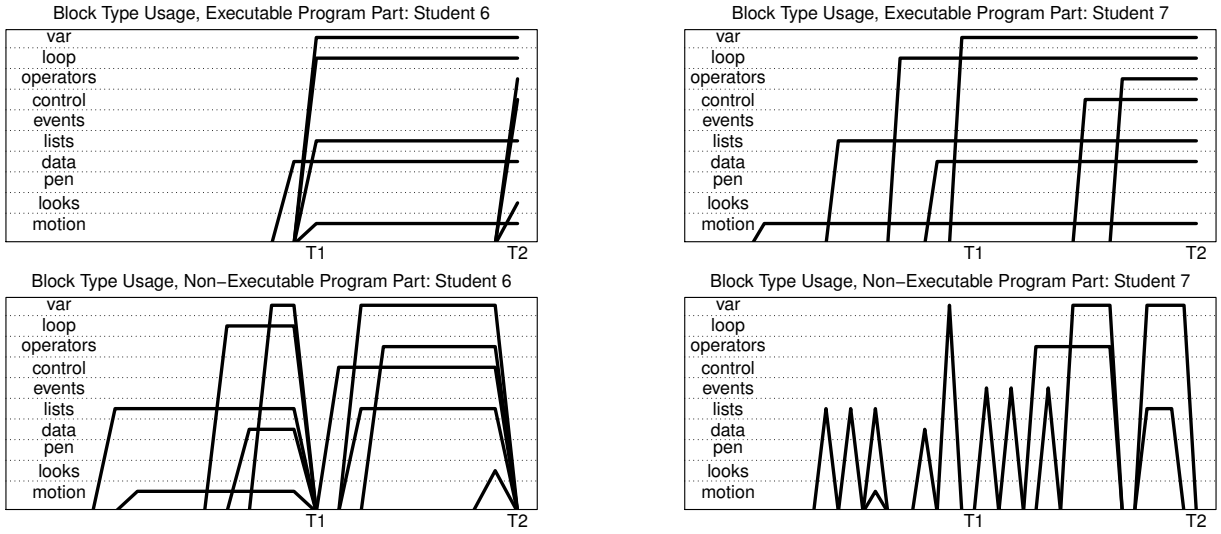
Programming has already been subject to analysis with *learning analytics* and *educational data mining*, collecting and analyzing data in educational programming processes in order to improve pedagogy, course and tool design [9]. In this notion, we propose to record sequences of program construction and interactions in the Scratch programming environment in order to gather measurable evidence of patterns and strategies used during the construction of Scratch programs. The aim is to quantify programming interactions for a fine-grained assessment of programming skills.

II. Trial Setup and Measurement

Scratch is occasionally used in Austrian upper secondary schools to introduce programming. We conducted a trial with eight school students at the age of 16 to 18. The students had to solve a given problem in the block-based programming environment *Scratch 2*. The *Scratch 3* online editor was not available at that time. The problem, drawing a trajectory by iterating through a list, is divided in two tasks (*T1*, *T2*) and requires 14 blocks and 7 main block types to be solved. It includes the concepts of loop, conditional, comparison, variable and list. The students' mean completion time was 14.80 ± 6.50 minutes. We observed the students and assigned a programming skill level in the integer range 1 to 4.

The programming interactions were recorded by logging mouse and keyboard events, and screenshots on each mouse click. Logged timestamps and mouse coordinates make it possible to relate the events to specific programming actions. During the trials, a mean number of 322 ± 170 programming interactions (mouse and keyboard events) were recorded.

The recorded click events were semi-automatically processed. Each event was categorized regarding: the area of interaction; the interacted block type; start and end area for drag events dealing with program blocks; type of program change (addition or removal to different program parts). All but the type of program change was automatically categorized. We manually extracted the used block types after each event, and computed the change rates by dividing the block type changes by the number of programming actions. For detailed analysis, the block types are categorized more fine-grained compared to the main block categories of *Scratch 2* (*loop* is separated from *control*, explicit variable usage *var* and *lists* is separated from *data*). Two sequences of block type usage can be seen in Figure 1. Student 6 mostly programs in the non-executable part (bottom left). Student 7 develops the main program (top right) and tests single blocks and changes in the non-executable part (bottom right).



(a) Student 6, 19 total programming actions

(b) Student 7, 36 total programming actions

Fig. 1: Sequence of block types usage for the program construction of two students. On the x-axis, the sequence of programming actions is plotted. On the y-axis, the used block types are shown. The labels *T1* and *T2* denote the programming action that solve the respective task. Student 6 follows a **Subprogram** strategy, programming in the non-executable part (bottom left) before adding multiple blocks to the main program (top left) at once, solving the respective task.

III. Trial Results

Correlation analysis employing Spearman rank correlation shows that three interaction metrics exhibit a strong correlation to the assigned programming skill level of the students, significant at $p < 0.05$. A lower trial completion time (-0.87), and a lower fraction of block delete actions (-0.93) are correlated to higher skill level. Higher geometric mean change rates of used block types, incorporating the change rates of executable and non-executable program parts and representing program changes that alter the used block types, are correlated to higher skill level (0.93). A caveat is that the latter correlation might be specific to the crafted example problem requiring different block types to be solved. Still the approach can provide a fine-grained assessment of programming skills during a programming task by observing the program construction sequence.

Based on heuristics derived from analyzing the students' program construction, we assigned four patterns and strategies to the students (**Trial & Error** programming, **Unfamiliarity** with the environment, **Late Abstraction** by beginning to program on explicit rather than general cases, **Subprogram** construction). Factorizing the 55 interaction metrics into three levels [*low*, *medium*, *high*] and using association rules mining, we obtained a set of metrics and factor levels for each pattern. The cardinality of the sets range from 8 to 49, with only 0 to 3 intersections. This preliminary result shows the possibility to identify patterns and strategies with interaction metrics.

IV. Conclusion and Future Work

In this contribution, we report on a trial conducted with upper secondary school students solving a given problem in the block-based programming environment Scratch. Their programming interactions were recorded by screenshot, mouse and keyboard logging with the aim to asses the programming skills with measurable interaction metrics. Some interaction metrics show a strong, significant correlation to the students' assigned programming skill level. Sets of metrics have been ascribed to patterns and strategies. These first results indicate that a fine-grained identification and assessment of programming skills is possible with program construction metrics. As future work, we plan to investigate more students and example problems, incorporating Java programmers. We plan to use program construction patterns to adjust teaching instructions and improve the acquisition of programming skills.

References

[1] A. Luxton-Reilly, "Learning to program is easy," in *ITiCSE '16*. New York, USA: ACM, 2016, pp. 284–289.

[2] S. Papert, *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, Inc., New York, USA, 1980.

[3] C. Watson and F. W. Li, "Failure rates in introductory programming revisited," in *ITiCSE '14*. New York, USA: ACM, 2014, pp. 39–44.

[4] J. M. Wing, "Computational Thinking," *Commun. ACM*, vol. 49, no. 3, pp. 33–35, Mar. 2006.

[5] MIT Media Lab, "Scratch," https://scratch.mit.edu/, 2019, [Online; accessed 01/2019].

[6] M. Armoni, O. Meerbaum-Salant, and M. Ben-Ari, "From Scratch to "real" programming," *ACM TOCE*, vol. 14, no. 4, pp. 1–15, 2015.

[7] F. Hermans and E. Aivaloglou, "Do code smells hamper novice programming? A controlled experiment on Scratch programs," *IEEE ICPC*, vol. July, pp. 1–10, 2016.

[8] O. Meerbaum-Salant, M. Armoni, and M. Ben-Ari, "Habits of programming in scratch," *ITiCSE '11*, p. 168, 2011.

[9] P. Ihantola, A. Vihavainen, A. Ahadi, M. Butler, J. Börstler, S. H. Edwards, E. Isohanni, A. Korhonen, A. Petersen, K. Rivers, M. Á. Rubio, J. Sheard, B. Skupas, J. Spacco, C. Szabo, and D. Toll, "Educational Data Mining and Learning Analytics in Programming : Literature Review and Case Studies," *ITiCSE WGR'16*, pp. 41–63, 2015.