Analysing Students' Scratch Programs and Addressing Issues using Elementary Patterns

Kashif Amanullah
Department of Computer Science
University of Canterbury
Christchurch, New Zealand
kashif.amanullah@pg.canterbury.ac.nz

Tim Bell
Department of Computer Science
University of Canterbury
Christchurch, New Zealand
tim.bell@canterbury.ac.nz

***Abstract*—In this Work in Progress paper in the Research Category we report on existing concerns about Scratch programming, and introduce patterns as a possible solution. Scratch is a popular language for introducing students to programming, but there is a concern that the students might not be exposed to all the key elements of programming when the development environment tempts them to explore elements such as the range of sprites available. We propose the use of programming patterns as a measure of the sophistication of student work. To understand the importance of patterns we report on our initial work that analyzes a large number of projects from the public Scratch repository to evaluate how extensively the basic patterns appear in student work. This can help inform the improvement of teaching methods to include use of broader range of patterns.**

***Index Terms*—programming patterns; Scratch; primary school students**

I. Introduction

Block-based programming languages such as Scratch have become popular in early school curricula. With this there is a risk that teaching might focus on surface features of the language, rather than deeper programming concepts. Programming is a challenging skill, and a progression of difficulty is appropriate. This raises the question of how learning should progress, to avoid students being stuck using only relatively simple programming concepts, or simply exploring cosmetic elements of the language. They might be building larger and larger programs, but still not learning how they can apply their programming to different computational problems.

This Research Work in Progress paper presents an investigation of issues found in Scratch programs shared by users on the Scratch public repository. We use "elementary patterns" (patterns that are suitable for novices to help them learn fundamental programming skills [1]) as a lens to identify the sophistication of the programming being done by students in a large sample of Scratch projects. Unsurprisingly, beginner programmers tend to use only very simple programming constructs. Based on this, we propose an approach to ensure that students are able to progress to a wider range of techniques to broaden their programming skill.

Scratch is by far the most popular block-based programming language. It does an excellent job as an introductory medium to the world of programming, partly because it has a very strong community. Still, there are concerns among the research community that Scratch could lead to unwanted programming practices in novice programmers.

In Section II we review issues from the literature around programming in Scratch. We then introduce patterns and how they can help in solving these issues in Section III. In Section IV we give an analysis of Scratch projects that shows how little these patterns are being used, and use this to inform what is needed to extend student's programming.

II. Issues with Scratch programs

There is no doubt about the utility and reach of Scratch, but it is not without its drawbacks. A number of studies raise concerns after looking into code smells, code quality issues, and bad programming habits. Aivaloglou *et.al* [2] harvested a large number of projects from Scratch public repository into a publicly accessible database. They used Dr. Scratch [3] to report the programming skills used in the Scratch projects based on seven dimensions of computational thinking. Aivaloglou and Hermans [4] further used this dataset to perform an analysis based on software metrics. The results of the study substantiate our concerns that children might be becoming familiar with the use of Scratch, but their programming is not exercising deeper concepts. For example, only 7.48% of projects used 5 or more variables; 78% of projects had a cyclomatic complexity of 1 (which means no control or decision points), and 13.08% of projects had a cyclomatic complexity of 2 (exactly 1 decision point). Techapalokul [5] analyzed a million projects submitted to the Scratch public repository and came up with a number of bad smells in the code. Moreno and Robles [6] developed two plugins to automatically detect two bad programming habits they found to be regular with high school student' code. They downloaded 100 Scratch projects for analysis, which verified their observations. Meerbaum-Salant et al. [7] worked with 9th grade middle school students using Scratch for one semester and identified two poor programming habits with their codes.

Table I summarizes some key problems that these authors have found in students' Scratch programs, along with some suggested solutions that have been reported in the literature.

III. Patterns for teaching programming

Despite significant research in the domain, there is a lack of formal or viable solutions to many of the problems highlighted

978-1-5386-1174-6/18/$31.00 ©2018 IEEE

TABLE I
A SUMMARY OF ISSUES WITH SCRATCH PROGRAMMING

Problem	Description	Suggested solutions
Bottom-up Programming	This is when the program is constructed by using primitive elements, which are combined to build up a more and more complex program. Instead of approaching a problem on algorithmic and design level, novices tend to drag all seemingly appropriate blocks into the solution space and then start building the solution [7].	Scratch should be used in the same way as other programming languages to teach good programming habits infused by solid teaching [7]
Extremely Fine-Grained Programming	This is when a program is constructed starting from a high level description and broken down to the primitives of the programming language. Novices also take the top-down approach to the extreme [5], [7]. Novices tend to break the problem into small pieces devoid of any logical coherency.	The teaching methodology and textbooks could play a vital role in encouraging the use of complex programming constructs as opposed to Extremely Fine-Grained Programming [7]
Poor Object Naming	Students do not change the automatic name provided by the environment (e.g. cat1, button2, etc.), which makes the code unreadable and difficult to debug, makes objects in large programs extremely difficult to identify, and slows down the overall programming process [5], [6]	Scratch doesn't enforce the deliberate naming of a new variable, and a name is assigned automatically. A change in this feature could improve the situation [5], [6]
Code Repetition/ Duplicated Code	There is a large percentage of code cloning and duplicated code in the Scratch projects repository [4]–[6]	Code repetition could be reduced by teaching abstraction and modularization [6]
Unreachable Code/Dead Code	A significant percentage of scripts contain dead code (uninvoked procedures, unmatched broadcast-receive messages, uninvoked code, empty event scripts, unused variables) [4], [5]	Scratch programmers could benefit from: a separate workspace to store unused blocks, sharing projects, and sharing functionality (in the form of a library). This would help novices a great deal to get started [4]. Scratch also lacks support to identify dead/unused code [5]
Long Scripts	A large number of projects exhibit this code smell [4]. It is a sign of "inadequate decomposition" and can "hinder code readability" [5]	The quality of code can be improved by implementing support to identify this and other code smells and by training programmers to avoid these code smells [5]
Broad Variable Scope	In Scratch all variables have global scope by default, and are visible between sprites. Having so many global variables means that the programmer cannot tell if it is "private" to a sprite, and the menus for choosing a variable become cluttered with irrelevant options [5].	The default behavior of Scratch is to have global variable scope [5], [6] which can't be changed. Making it possible to change scope might encourage appropriate scope, otherwise students need to get it right before the program is complete [6]

in Table I. The aforementioned problems call for an approach that instills good programming habits, trains students to avoid bad programming style, and opens them to a range of programming techniques. Programming patterns are a promising way to achieve these goals, and we explore how these could address existing problems in Scratch programs. Patterns have been applied successfully for teaching tertiary students [1], [8], but haven't been explored for younger students. We propose the application of elementary patterns at primary school level. Although elementary patterns were primarily designed for older students, they can be adapted to teach good programming practices to children. An advantage of using patterns as an approach is that it teaches problem solving independently from whichever programming language is being used. Using a teaching pedagogy based on patterns can help achieve the following benefits:

- It can show novices' worked programming solutions to known problems and will show them how to structure the program and how to write code using good style (since reading code is an important step to learning to write it [9].)
- Teaching good programming practices will help reduce code smells, particularly those mentioned in Table I.
- It can prepare children at an early age to face the complex programming challenges that they will encounter in the future in a more confident way [10].
- Patterns work in most programming languages that students are likely to encounter, so as they move to new programming languages they will be able to generalize what they have learned.

Elementary patterns lend themselves as a natural choice to teach programming based on the issues identified above with block-based programming. Bottom-up programming, and extremely fine-grained programming highlight the issue of novices having a lack of understanding with algorithmic design, which can be mitigated with the use of elementary patterns as they become aware of the common approaches that can be used for problem solving.

Similarly, code repetition, dead code, and large scripts are issues raised by many researchers. Elementary patterns can also be helpful here because they are effective solutions to common programming problems, and using those solutions will naturally reduce the number of blocks used. Also, functions are a good way to organize and reduce code repetition. Although some curricula recommend introducing functions later, perhaps because they are perceived to be a complex topic not suitable for young children, in Scratch we can teach novices a simple form of decomposition that functions offer either with the "More Blocks" feature, or by using message passing, broadcast, and custom blocks in Scratch.

A list of some of the most relevant elementary patterns is given in Table II. This list was created by merging patterns from [11] and [12] that can be implemented in Scratch. Some useful patterns are also given in [13], which targets good

TABLE II
PATTERNS SELECTED FOR ANALYSIS OF SCRATCH PROGRAMS

Loop Patterns

Process All Items Process all items in a collection (such as a list or file)

Linear Search Loop over a collection and stop when a condition is met

Guarded Linear Search Loop over a collection, stop when a condition is met and provide an alternative action if the condition is not met

Loop and a Half Loop over a collection until a sentinel is reached (the number of items in the collection is not known in advance)

Polling Loop Ask the user to enter a value, then loop until the user enters a valid value

Extreme Values Loop over a collection to find extreme values in a collection (e.g. maximum or minimum); initialize the extreme value to the first value in the collection and replace it if a better candidate is found

Selection Patterns

Whether or Not Use an `if` statement without an `else` part to test a condition; there are no other actions to do instead of this one

Alternative Action Use an `if` statement with an `else` part; exactly one of the two actions is appropriate based on a condition

Unrelated Choice Executing several actions that each have associated conditions; each condition/action pair is decided independently

Independent Choice Use nested `if` statements when only one action must be taken and the action depends on several independent factors

programming practices and could address many problems found with Scratch programming, although these patterns aren't covered in this paper; for example, one of Bergin's patterns is "Consistent Naming", which would be useful for students to follow to overcome the issue of poor object naming highlighted in Table I.

IV. Analysis of Scratch programs

As an initial step towards thinking about patterns for young novices, we have evaluated what kind of patterns they already use. The online public Scratch repository has a huge number of projects that are accessible to all, so by analyzing these programs, we can get an idea of the coverage of elementary patterns amongst Scratch users. This will help us see the breadth of techniques that students are using for their programming.

Scratch has a very large and active community and the number of projects shared increases every day. At the time of writing there were 30,868,915 projects shared on-line. We have sampled from these projects to keep download and processing time reasonable; the results reported here are based on 212,250 projects that were selected to be representative of the range of projects that students have made available. Projects were selected by taking every project in a numeric sequence, which essentially selects projects from a date range. The projects sampled here were mainly from one weekday of activity in March 2016 and April 2018 each.

We have only analyzed programs written in Scratch 1.4 and Scratch 2.0, which forms the major proportion of shared projects. Scratch 1.4 was officially released on July 19, 2009, which is only two and a half years after the first version of Scratch (Scratch 1.0, released on January 8, 2007). Scratch 2.0 was released on May 9, 2013, and has been in use since then; it has the largest number of projects of any version.

We have analyzed the 212,250 Scratch programs to gather statistics about students' use of the language, with the main focus being on finding patterns. Python software was written to run through each file and find the frequency of elementary patterns, which are shown in Table II. The patterns are divided into the two main categories: Loop patterns and Selection patterns. Most loop patterns work on a collection, which is significant because there is only one type of collection provided in Scratch, the list. Being able to perform certain actions (add items, search items, remove items) on a collection, and the ability to perform decision making in complex programming situations is a sign of a good programmer. These skills indicate an understanding of control and flow structures, problem solving, and logical thinking, which is exactly what loop patterns try to explore.

Table III shows our analysis of the use of various command blocks and features in the sample Scratch programs, as well as patterns. The analysis shows that less than three percent of the programs use a list, which means that less than 3% of projects are likely to use Loop patterns.

The "repeat", "forever", and "repeat until" blocks in Scratch represent an important programming construct i.e. loops. Good understanding of using loops is one of the fundamental requirements of using the full power of programming. We can think of the "repeat" block as a simplified version of a "for loop" and the "forever" and the "repeat until" blocks are a special case of a "while loop".

The usage of repeat and forever loops is around 19.97% and 33.11% respectively, which is a high proportion considering that young students form a major part of Scratch users, and shows familiarity with the general idea of a loop. However, these are very simple constructs that appear in introductory Scratch tutorials as a common pattern. The "forever" block is frequently used as either a continuous animation, or a busy polling loop, where it might be testing repeatedly for a condition such as a sprite reaching the edge of the screen. In many programming languages this would be very inefficient, but Scratch seems to deal with these by having relatively large time delays on each loop, which means that other forever loops can also be polling for a condition without heavy CPU usage. This behavior is unique to Scratch and unsuited for pattern-based approaches, which might be one of the reasons for low usage of loops in more generally applicable patterns, and inelegant solutions involving forever loops. The "repeat until" loop (the only loop with a Boolean expression) is used in only 7.06% of the projects, suggesting that relatively few students are able to use loops in combination with conditions for termination.

The List is the only data structure provided in Scratch that can hold a collection of data. The ability to manipulate the list becomes a vital skill set in a programmer's toolkit. However,

TABLE III
PATTERNS FOUND IN ANALYSIS OF SCRATCH PROGRAMS

Pattern	Percentage
Repeat	19.97%
Forever	33.11%
Repeat Until	7.06%
Search	3.83%
List	2.81%
Process All Items Pattern	1.17%
Linear Search Pattern	0.02%
Loop and a Half	0.20%
Polling Loop Pattern	0.48%
Whether or Not Pattern	24.83%
Alternative Action Pattern	8.53%
Nested if/else	0.68%

less than 3% of the projects used lists.

Process All Items is a pattern which shows the ability of a programmer to traverse and process all items in a collection using loops. Only 1.17% of the Scratch programs showed this pattern.

Search is an essential feature of many computer based applications, and it is key for a programmer to know how to implement it. Advanced searching algorithms can be quite complicated, but a basic search is relatively simple, although could still be a daunting task for young programmers. Therefore not surprisingly, when we looked for the signature of a basic search (a "repeat"/"forever" block containing an "if/else" block), only 3.83% of the projects contained it. Of course, the signature doesn't mean that the program is using a linear search; the use of the actual "Linear Search" pattern (traversing a list using a loop and finding a specific item) is significantly lower than this number (only 0.02%).

The "Loop and a Half" pattern is a good pattern to evaluate programming skills, as it looks for the use of multiple structures in a program. A correct use of this pattern requires a good understanding of loops ("forever" block), collections ("list"), user input ("ask"), control structures ("if/else"), and using most these in combination. This pattern is only found in 0.20% projects.

In the "Polling Loop" pattern, the loop runs until the user enters a valid value. This functionality can be achieved using "repeat until" block in Scratch. Usage of this pattern is found in only 0.48% of the projects. The "Polling Loop" pattern can also be written using the "Loop and a Half" pattern [11], therefore the occurrence of 0.15% for the "Loop and a Half" can also be considered as contributing to the "Polling Loop" pattern.

"Whether or Not" and "Alternative Action" patterns correspond to the "if" construct of programming languages. Selection and decision making is a big part of programming, and is often one of the first command structures taught when starting programming. Although it is very basic, for young programmers it can still involve significant comprehension effort on their part, particularly since it involves Boolean logic. The "Whether or Not" pattern involves testing a condition, and "Alternative Action" adds a course of action in case that condition is not satisfied. All this requires programmers to construct a proper condition, decide on which logical operators to choose, and what alternative route to take in case the first condition is not met. All of this can be a challenging task for young learners. The results show that the coverage of the "Whether or Not" pattern is 24.83%, which shows that a reasonable number of projects are indicating familiarity with the "if" condition.

The "Alternative Action" pattern, on the other hand, was found in only 8.53% of the projects, showing a preference for using the "if" condition, and perhaps a lack of experience with "if/else".

The "Unrelated Choice" pattern and "Independent Choice" pattern both examine the use of nested "if/else". The use of a nested "if/else" shows the ability to handle complex conditions. It is significantly more complicated than using simple "if/else" structures since it checks for multiple conditions to make a decision. This is also confirmed from the results as the use of nested "if/else" is only 0.68%.

One of the major reasons for this exercise was to highlight the issues with Scratch programming and to argue that elementary patterns could be a useful solution to many of these problems. We have observed through the analysis of Scratch programs available on Scratch public repository, and based on the numbers retrieved, that Scratch programs do show most of the problems raised in literature. Elementary patterns, due to their inherent nature of advocating problem solving and developing logical thinking, present themselves as a good solution. It is also important to note that some of these patterns might be thought of as too complicated for young students, but as students are exposed to information earlier and earlier, they may have the opportunity to develop cognitive skills to understand and use these patterns.

V. CONCLUSIONS

Scratch is widely used as an introductory programming tool, but it could be giving rise to some unintentional issues. We have thrown light on some of these issues taken from literature and proposed a solution based on elementary patterns. The analysis of programs from the Scratch repository strengthens our stance that these patterns are being under-used despite having all the ingredients to be a good choice to develop problem solving and logical thinking skills in young programmers.

For future work we are doing user-based patterns usage and the progression of programming skills based on the analysis of Scratch projects. To demonstrate that patterns are important and fairly common but still not taught, an analysis of materials for teaching programming (books, online resources, etc.) will also be conducted. Also, we will be conducting studies with young students to show the impact of introducing patterns in their programming lessons.

REFERENCES

[1] M. J. Clancy and M. C. Linn, "Patterns and pedagogy," in *The Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '99. New York, NY, USA: ACM, 1999, pp. 37–42. [Online]. Available: http://doi.acm.org/10.1145/299649.299673

[2] E. Aivaloglou, F. Hermans, J. Moreno-León, and G. Robles, "A dataset of Scratch programs: Scraped, shaped and scored," in *Proceedings of the 14th International Conference on Mining Software Repositories*, ser. MSR '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 511–514. [Online]. Available: https://doi.org/10.1109/MSR.2017.45

[3] J. Moreno-León, G. Robles *et al.*, "Analyze your Scratch projects with Dr. Scratch and assess your computational thinking skills," in *Scratch Conference*, 2015, pp. 12–15.

[4] E. Aivaloglou and F. Hermans, "How kids code and how we know: An exploratory study on the Scratch repository," in *Proceedings of the 2016 ACM Conference on International Computing Education Research*, ser. ICER '16. New York, NY, USA: ACM, 2016, pp. 53–61. [Online]. Available: http://doi.acm.org/10.1145/2960310.2960325

[5] P. Techapalokul, "Sniffing through millions of blocks for bad smells," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '17. New York, NY, USA: ACM, 2017, pp. 781–782. [Online]. Available: http://doi.acm.org/10.1145/3017680.3022450

[6] J. Moreno and G. Robles, "Automatic detection of bad programming habits in Scratch: A preliminary study," in *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, Oct 2014, pp. 1–4.

[7] O. Meerbaum-Salant, M. Armoni, and M. Ben-Ari, "Habits of programming in Scratch," in *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '11. New York, NY, USA: ACM, 2011, pp. 168–172. [Online]. Available: http://doi.acm.org/10.1145/1999747.1999796

[8] A. V. de Aquino Leal and D. J. Ferreira, "Learning programming patterns using games," *International Journal of Information and Communication Technology Education (IJICTE)*, vol. 12, no. 2, pp. 23–34, 2016.

[9] R. Lister, E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, R. McCartney, J. E. Moström, K. Sanders, O. Seppälä, B. Simon, and L. Thomas, "A multi-national study of reading and tracing skills in novice programmers," in *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, ser. ITiCSE-WGR '04. New York, NY, USA: ACM, 2004, pp. 119–150. [Online]. Available: http://doi.acm.org/10.1145/1044550.1041673

[10] L. Seiter and B. Foreman, "Modeling the learning progressions of computational thinking of primary grade students," in *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research*, ser. ICER '13. New York, NY, USA: ACM, 2013, pp. 59–66. [Online]. Available: http://doi.acm.org/10.1145/2493394.2493403

[11] O. Astrachan and E. Wallingford, "Loop patterns," in *Proc. Fifth Pattern Languages of Programs Conference*, Allerton Park, Illinois, 1998.

[12] J. Bergin, "Patterns for selection version 4," https://csis.pace.edu/~bergin/patterns/, 1999.

[13] ——, "Coding at the lowest level: Coding patterns for java beginners," in *EuroPLoP*, 2001, pp. 251–286.