Analysis of Progression of Scratch Users based on their Use of 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*—Block based programming languages are a popular way to introduce computer programming to young students. However, concerns have been raised that students may not be learning important programming skills. Previous analyses of Scratch projects online have revealed that the use of important computational elements such as variables, *if* statements and controlled loops is extremely low, and introducing elementary patterns to the teaching sequence emerged as a potential solution. Although students might have success in the Scratch environment with a limited vocabulary of constructs, it has been pointed out in the literature that the transition process to a different programming language might be difficult. A concern with prior studies of a static analysis of uploaded Scratch projects is that they might be biased towards one-off or brief encounters with programming, where students have a short experience, upload a simple program, and share no further work. We address this concern in this study by measuring the progress of Scratch users over the years, particularly through their use of elementary patterns. Our study has identified all projects submitted by 34,832 users in the Scratch community, and for each user we have compared the skills demonstrated in the first half with those in the second half of their projects. There were no clear signs of any progression, and even some indications of negative progression, and even after more than a year of programming, for most users we found that the use of key elements of programming was still low.**

***Index Terms*—introductory programming, patterns, Scratch, remix**

I. Introduction

Block-based programming languages have become very popular in recent years, and are widely used in early computer science education. Their role has been instrumental in introducing computer programming to children [1], [2]. Scratch is a clear leader [3] in block based languages, with a huge community of users and many teaching resources written for it.

A lot has been written about how block-based languages are effective and useful, but there are also issues with block-based languages highlighted in the literature, which need to be addressed [4]–[6]. Some of the issues noted include a focus on bottom-up programming as students discover and experiment with blocks, poor naming of objects because students use default names, code repetition and long scripts due to lack of modularisation or decomposition, and the "low floor" of the language that makes it easy for students to write simple programs, but encourages them to experiment in an ad hoc manner, without discovering the more powerful constructs. Concerns have been raised that students are not learning a full range of programming skills due to these issues. Of course, this isn't necessarily a weakness if the language is being used primarily as a first experience in programming; however, it is also used for more senior classes [7], [8], so it is useful to understand how students can progress in this environment.

Learning computer programming is a gradual process, and it requires becoming proficient with basic programming constructs. A study with over 200,000 Scratch projects from a public Scratch repository confirmed that the use of important computational elements such as variables, *if* statements and controlled loops is extremely low. Based on these results, elementary patterns were proposed as a potential solution [9]. The elementary patterns [10] chosen (see Table I) reflect common simple programming constructs needed to access the full capabilities of computation. They are adapted from the work of Bergin [11], [12], and Astrachan and Wallingford [13] with the "Search" pattern added to allow for a simpler form of linear search that doesn't require a collection.

Sound programming skills are independent of the programming tool being used and should be transferable to any other programming language. If students are taught programming *skills* instead of programming *tools*, the challenge of transition to text-based languages or any other block-based language, which has been highlighted in the literature as problematic for many students [14], will be much smoother. An elementary pattern-based teaching approach, which would expose students to a variety of programming structures, is a promising step in this direction. Dorling and White [15] emphasized the use of a good teaching pedagogy to teach programming by using both block-based and text-based in combination to facilitate the process of transition.

Prior studies of student work show extremely low usage of useful constructs by students (e.g. [16]), but the statistics reported are taken over all projects developed by users, whether it is their first project or if they have been programming for months. This raises the possibility of the results being unduly skewed by students who have had only one short period of programming, which one would naturally expect to have simpler constructs. Scaffidi and Chambers [17] measured the

TABLE I
PATTERNS SELECTED FOR ANALYSIS OF SCRATCH PROGRAMS

Loop Patterns

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

Search Loop and stop when a condition is met

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

Guarded Linear Search (gls) 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 (laah) Loop over a collection until a sentinel is reached (the number of items in the collection is not known in advance)

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

Selection Patterns

Whether or Not (if) 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 (ifelse) Use an `if` statement with an `else` part; exactly one of the two actions is appropriate based on a condition

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

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

skill progression of users using four models and found negative results for technical skills progression, although some of the findings were refuted by Matias et al. [18]. The main purpose of this study is to investigate the *progression* of Scratch users i.e. how long have they spent on Scratch, how many projects have they created and shared, and how much progress they have made over a span of time. We use elementary patterns as a basis for the analysis, fetching all projects associated with each user from the Scratch public repository, and dividing a user's set of projects into two halves (their earlier projects, and their later ones), and then counting the number of each pattern used in these, and the way the usage of patterns progresses over time. This will enable us to find out if students using Scratch develop a broader repertoire of programming idioms, or if the environment and typical pedagogy continues to encourage them to remain using only simple patterns.

It would be reasonable to expect students to make progress when they work with something for a longer period of time. This research aims to collect evidence to verify whether this is true in the case of Scratch programming, since we are interested in whether or not programming in Scratch helps learn sophisticated programming concepts. Elementary patterns are a collection of relevant sophisticated concepts, and we compare the use of patterns in the first and second half of the projects that each user uploads.

The rest of the paper will focus on the analysis of the projects and discussion of the results. We begin by looking at how the use of patterns changes over time, and then consider the differences between experienced and novice users.

TABLE II
FIVE NUMBER SUMMARY FOR PROJECTS AND SPAN

	min	Q1	median	Q3	max
Projects	1	4	9	23	8945
Span (days)	0	21	78	185	4032

II. ANALYSIS, EVALUATION, AND RESULTS

This project focuses on two sets of projects posted by each user to the online Scratch repository, so that we could evaluate the progress over a user's span of engagement with the system. Of course, because Scratch can be used offline, it is possible that their project list is not complete, but we are working with a very large sample size to avoid exceptional cases having any significant impact.

We started our analysis by extracting users' information from a collection of 373,497 Scratch projects that had been downloaded from the online Scratch repository. We processed these 373,497 projects and extracted the details of 34,832 users that had one or more projects. We can notice there are a lot more projects than users. Of course, many users have more than one project, but also the projects downloaded via the Scratch repository API include projects which might not be shared online anymore, so a large number of projects were identified were not accessible when processed via the API. The extracted information included the number of projects shared by each user, ids of the users, and the span (time spent in days between first and the last projects). Based on this information, we evaluated the relevant projects from this sample of users to generate the statistics reported below.

Table II provides some useful insights about projects shared by users. 25% of the users have 4 or fewer projects, and half of the users have 9 or fewer projects. The maximum value of 8,945 projects is an outlier, and not representative of the population (less the 0.3% have more then 1000 projects); the users with a very high number of projects appear to be a shared account that multiple students were using). 25% of the users have shared projects over a span of 21 days or less, and a further 25% spent 21 to 78 days working with Scratch; given that these periods represent 3 weeks to two and half months approximately, this work may represent a one-off series of lessons in class, or an event such as a summer camp.

Table III also shows some more interesting facts about Scratch users. 12.64% of the users had a zero span, which means they worked on a single project just once, and never returned. The mean, and median number of projects per user is 35.89, and 9 respectively and 54.53% of users have 10 projects or fewer. The median is more representative in this scenario, as few outliers would have an undue influence on the mean. 29.51% of users have a span of less than a month working with Scratch, which seems like insufficient time for proper programming concepts to sink in, especially if the number of projects is also lower, although we acknowledge that a shorter

TABLE III
SUMMARY STATISTICS ABOUT SCRATCH USERS

Number of users with zero span	4,482 (12.64%)
Mean number of projects for 34,832 users	35.89
Number of users with 1 project	3597 (10.33%)
Number of users with over 5 projects	22,605 (64.90%)
Number of users with span of less than a month	10,278 (29.51%)
Number of users with span of over a month	24,410 (70.08%)
Number of users with span of over 3 months	16,540 (47.49%)
Mean span for 34,832 users	168.71 days

experience can give students a view of what programming is about, even if they don't develop expertise.

One concern with projects uploaded to the Scratch community is that students can "remix" others' projects, so the programs associated with a user may not be entirely their own work. This can be used positively to give students the opportunity to view model programs, and then modify them as part of their learning [19], which in principle could expose them to new structures and patterns. However, we have observed that often the modifications are not around learning programming (such as changing a backdrop or sprite), and in some cases almost no modification is made at all [20]. Since, we are working with all projects for each user in our sample, the remixed projects will not have a big impact on the results reported here as the remixing percentage is usually between 0-25% [21].

To investigate the possibility that users have created a larger number of projects in a short span of time (and therefore gained considerable experience), the relationship between the number of projects and the span of time over which users were submitting projects is shown in the form of a scatter plot in Figure 1. This shows a concentration of spans under 3 to 4 years, which would appear to match anecdotal evidence that students grow out of block-based languages after a while, either because they see it as too childish, or because the effort of dragging and dropping elements for sophisticated programs can be more time-consuming than typing in a text-based language. We see this as being akin to a child learning on a bicycle with training wheels and streamers on the handlebars — it's very exciting and empowering to start with, but after a while what was once novel becomes inconvenient and even embarrassing.

The range of the number of projects and spans show that some students are continually working on just a small number of projects, while others start many new ones. This might reflect different teachers' approaches (encouraging students to keep old versions of projects, or working on many different challenges), or it may reflect the personality of the students; we have encountered a number of students who do not want to keep old projects, as they want to improve it, and would rather not keep a record of their early work. 47.49% of the users have a span of over 3 months. The user group most suitable for analysing their progression as programmers will be a subset of this group as they would have spent a significant amount of time and have a greater number of projects over which they have developed their skills.

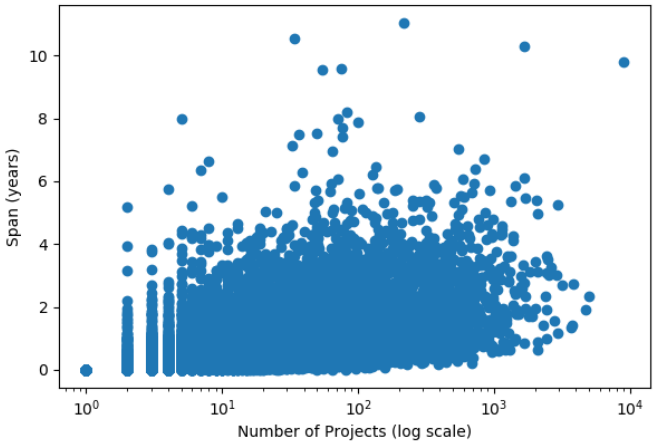

Fig. 1. Relationship between number of projects created and span over which they were submitted

Acquiring good programming skills requires spending time on the task. To analyse the development of users' use of programming constructs over time, we focus on the first and the last half of the each user's projects for all users having two or more projects. This has enabled us to discover the improvements made by users over time.

Table IV shows five number summaries for all users for each pattern in Table I for the first and the last half of the projects that each user has created, respectively. In addition, for comparison with previous studies, we also counted the number of variables used, and the use of the repeat, forever and until commands, and variables and lists. "Search" is a pattern similar to "Linear Search", with the only difference that "Linear Search" applies specifically to lists."Search" still shows use of sophisticated elements of programming considering lists might not be used by a large number of users (which is apparent in our analysis). Both the "Unrelated Choice" and "Independent Choice" patterns are very similar, using a nested if/else, so they have been combined in the results as "nestedif".

It is clear from the Table IV that maximum values are extreme outliers, as these are rare occurrences. These statistics reveal some intriguing results. The table list some of the most common and basic elements of programming that are fundamental towards achieving substantial programming skills and a stepping stone towards advanced programming skills. We can see that more than 50% of the users are rarely using common patterns at all. These numbers provide more evidence that supports concerns around how block based languages are taught, which has been highlighted in the literature [4]–[6], [16].

The concern is that teachers and students see Scratch as another program to learn, rather than a programming *language* that has all the key constructs needed to implement computational algorithms. This can result in students primarily using it as an animation tool, implementing simple programs based mainly on sequences of commands, and spending time

TABLE IV
FIVE NUMBER SUMMARY FOR EACH PATTERN FOR USERS' FIRST AND LAST PROJECTS

	min		Q1		median		Q3		max	
	first	last	first	last	first	last	first	last	first	last
Process All Items	0	0	0	0	0	0	0	0	2130	6431
Search	0	0	0	0	0	0	1	1	1336	650
Linear Search	0	0	0	0	0	0	0	0	119	55
Guarded Linear Search	0	0	0	0	0	0	0	0	119	55
Loop and a Half	0	0	0	0	0	0	0	0	84	32
Polling Loop	0	0	0	0	0	0	0	0	1333	955
if	0	0	0	0	4	5	23	24	22739	40215
ifelse	0	0	0	0	0	0	3	5	12639	30008
nestedif	0	0	0	0	0	0	6	6	68476	70442
repeat	0	0	1	0	4	2	15	12	32335	37270
forever	0	0	2	1	9	8	38	33	20518	33556
until	0	0	0	0	0	0	4	4	5994	7471
variables	0	0	0	0	3	3	15	15	38063	81194
list	0	0	0	0	0	0	0	0	9443	19702

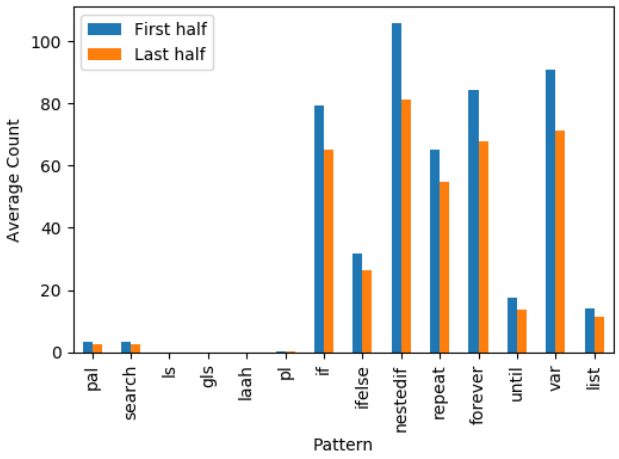

Fig. 2. Average number of patterns used by each user, for the first and last half of submissions

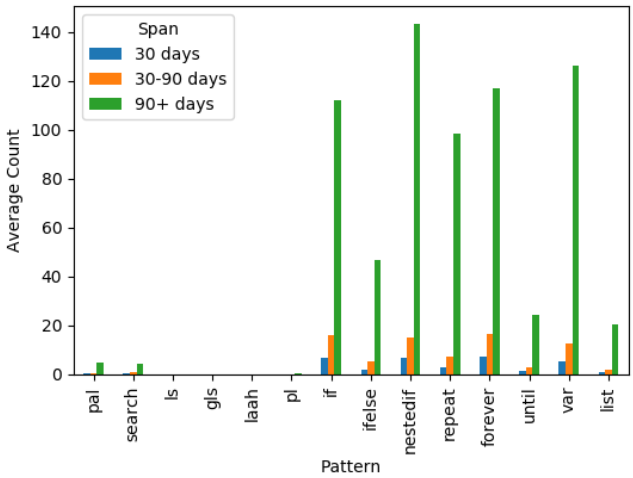

Fig. 3. Average number of each type of pattern used by users in the last half of projects divided on the basis of span

choosing or creating sprites rather than learning the constructs of programming.

The numbers in Table IV don't improve much for 75% of the users between the first and the last half of their projects, with all programming patterns not being used at all, including all of users not using *list* at all, even in the last half of projects. It is only towards the highest percentiles where we see improvements, with users using most of the constructs frequently. However, this is due to very few users using a large number of elements in some projects, and in fact this in itself can also be poor programming style; for example many users use hundreds of "if" statements that could have been replaced with a better construct such as a list with a linear search. The use of the "forever" statement is common in Scratch, as it often appears in teaching material as a way to have sprites check for a condition such as a collision using a busy poll; this structure works satisfactorily in Scratch because the loop's use of processing time is regulated, but such an approach would be unworkable in a lot of other languages.

Two points are clear from Table IV. One is that important programming elements are being under-used, and the other is that there is no clear evidence of progress over the span of time - in fact, the use of some elements/patterns is larger in the first half of the projects compared to the second. This could be an indication that users keep repeating the same practices and don't develop better programming skills. An optimistic view would be that they progress away from the online Scratch environment and explore more programming concepts in other environments.

In Figure 2 we show the *average* count of each pattern for all users for the first and the last half of their projects. This figure shows a slightly different picture than the five number summary i.e. the average use of each pattern is noticeably higher in the first half of the users' projects. This is mainly due to the averages being affected by a small number of users with high usage. Nevertheless, this is further evidence that users are not making much progress, and if anything it shows a negative trend. This is in line with prior results from Scaffidi and Chambers [17]. It is also apparent that some constructs

TABLE V
USER CATEGORIES BASED ON NUMBER OF PROJECTS

Number of projects	Number of users
1 project	3597 (10.33%)
2 to 5 projects	8630 (24.78%)
6 to 10 projects	6766 (19.42%)
11 to 30 projects	9033 (25.93%)
30 or more projects	6806 (19.54%)

TABLE VI
USER CATEGORIES BASED ON SPAN

Span	**Number of Users**
A week or less	6125 (17.58%)
Over a week and less than or equal to a month	4297 (12.34%)
Over a month and less than or equal to three months	7870 (22.59%)
Over three months and less than six months	7546 (21.66%)
Over six months	8994 (25.82%)

(ifelse, lists, and until) and programming patterns (Process All Items, Search, Linear Search, Guarded Linear Search, Loop and a Half, and Polling Loop) that could suggest more developed problem solving skills, are used very lightly. In fact, the elementary patterns are almost non-existent in this particular sample.

Figure 3 shows the use of each pattern in the last half of submissions, by dividing users into three categories based on the span (elapsed time spent between first and last projects). The purpose is to find out if spending more time leads to the use of more sophisticated programming elements. It is clear that the use of many programming elements is higher for users having a larger span, although we still see that the use of constructs connected to computational problem solving is very low.

III. NOVICE VS. EXPERIENCED USERS

The lack of the use of patterns in the previous section is heavily dominated by a large number of users who have a relatively short time learning programming. In this section we divide users into two different categories based on the number of projects and their span of submissions, to be able to look at the progress of those who have more experience compared with novices who have what might only be a surface experience of programming. Tables V and VI show different classifications of the users based on the number of projects they have created and the span of their work.

Based on the data from above two tables we divided users in two major groups: novice and experienced. The group of experienced users have more than 30 projects and a span of over 6 months, and the rest of the users fall into the novice group. The experienced group promises to contain more useful data regarding the patterns when compared with novice group due to the number of projects and length of time that they have been using Scratch. The total number of experienced users was 4988 (14.32%).

Analysis of five number summary revealed a noticeable increase in the use of important programming constructs (if, ifelse, nestedif, repeat, forever) for experienced users in the upper quartile range for both their first and last halves, but the use of elementary patterns was still near-zero for most of the users in the experienced category, and there were no direct signs of progression. Novice users' statistics were as might be expected, with only a slight increase of "if", "repeat", and "forever", but the rest of the elements remained at zero up to the third quartile of users.

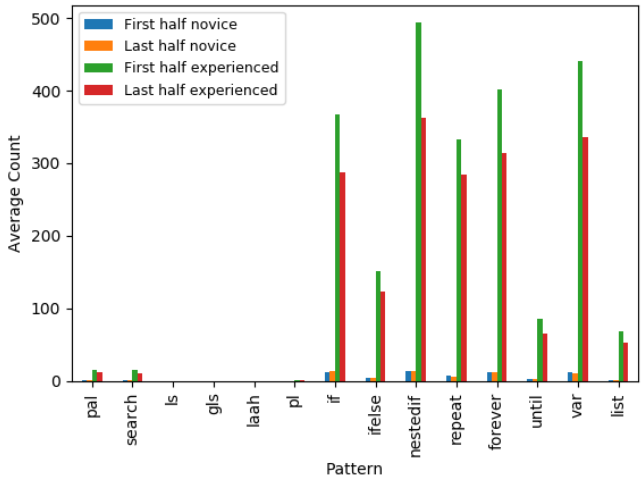

Fig. 4. Average number of patterns for the first and last halves of programs of the novice and experienced groups

Figure 4 shows the use of patterns in the first and last half of the projects for both novice and experienced groups; being averages per user, they are pulled up by very high usage by relatively few users. We see that the experienced users have a richer variety of patterns and constructs, even in the first half of their submitted projects. For example, the "experienced" group use variables, "if", "nestedif", "repeat", and "forever" many times more than the novice group in both halves. This indicates a different pedagogical approach, and seems to differentiate between offering students a short and less demanding experience, compared with a more rigorous and extended programme. In addition to the possibility that the "experienced" students were given better examples to work with right from the start, it could be that they had prior experience and are already motivated to do programming, resulting in them producing more projects and working over a longer period. Novice users (Figure 4) show some use of the constructs, but the total numbers are considerably lower than those of the experienced users.

IV. THREATS TO VALIDITY

This study raises as many questions as it answers, as we need to investigate the different ways that programs come about (e.g. some are remixes of other students' work), and the span of time that users engage with Scratch may reflect external factors such as taking a class or summer camp. In future work we will be investigating in more detail the difference between the usage of patterns in remixed projects

compared with original projects to evaluate the impact of remixes and to determine how many additions users make to existing projects, such as the number of lines of code (blocks) added, and the number of changes that are purely cosmetic.

The samples were all taken from the public on-line community, but students may also be using Scratch locally on their computers, so the results reported may not necessarily reflect the kind of work done by all users, since some may choose not to share their work, while some user accounts may be used by multiple students, such as a class working on the same project. It is also possible that students are downloading others' work, "remixing" it, and then uploading it, which will make it appear to be entirely authored by the student who remixed it. We will be conducting a survey in schools to investigate the workflows of online vs. offline Scratch work. It will be useful to see how well the projects shared online reflect what happens in a class that is taught offline.

V. Conclusion

Many Scratch users show no use of patterns and common programming constructs, even after they have had months of experience writing dozens of projects; only some of the most experienced students are likely to show much progress. This could be because Scratch isn't seen as a full programming tool (for example, it has been referred to as an "animation environment" [17]), and students end up using a range of simple features (such as changing the appearance on the screen) rather than engaging with the full power of computation that is possible in Scratch. It also likely reflects teaching material or approaches to lessons taken by their teachers, and the balance between offering students a short, positive experience, compared with a longer-term rigorous programme of study.

Despite this uncertainty, the results support the concerns that have been raised over the way students use block based programming, as important programming constructs and elementary patterns are not being used, even by many users who have had considerable experience in Scratch. The language itself, and other related block-based languages, are capable of supporting the concepts, so what may be needed to address the concerns is better pedagogy, through teaching resources that go into more depth, more time allocated to learning about programming, and upskilling instructors so they understand that the measure of a programmer's skill is not so much the length of programs, but the range of elements used in the programming.

References

[1] S. Y. Lye and J. H. L. Koh, "Review on teaching and learning of computational thinking through programming: What is next for k-12?" *Computers in Human Behavior*, vol. 41, pp. 51 – 61, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0747563214004634

[2] P. Hubwieser, M. N. Giannakos, M. Berges, T. Brinda, I. Diethelm, J. Magenheim, Y. Pal, J. Jackova, and E. Jasute, "A global snapshot of computer science education in k-12 schools," in *Proceedings of the 2015 ITiCSE on Working Group Reports*, ser. ITICSE-WGR '15. New York, NY, USA: ACM, 2015, pp. 65–83. [Online]. Available: http://doi.acm.org/10.1145/2858796.2858799

[3] T. R. Society, "After the reboot: computing education in uk schools," https://royalsociety.org/~/media/policy/projects/computing-education/computing-education-report.pdf, 2017.

[4] 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

[5] 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.

[6] 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

[7] T. Bell, H. Newton, P. Andreae, and A. Robins, "The introduction of computer Science to NZ high schools: An analysis of student work," in *Proceedings of the 7th Workshop in Primary and Secondary Computing Education — WiPSCE '12*, 2012, pp. 5–15.

[8] D. Thompson and T. Bell, "Adoption of new Computer Science high school standards by New Zealand teachers," in *The 8th Workshop in Primary and Secondary Computing Education (WiPSCE 2013)*, M. Knobelsdorf, R. Romeike, and M. E. Caspersen, Eds. Aarhus, Denmark: ACM, 2013. [Online]. Available: http://www.iitp.org.nz/files/wipsce-teachers-2013.pdf

[9] K. Amanullah and T. Bell, "Analysing students scratch programs and addressing issues using elementary patterns," in *2018 IEEE Frontiers in Education Conference (FIE)*, Oct 2018, pp. 1–5.

[10] E. Wallingford, "The elementary patterns homepage," https://www.cs.uni.edu/~wallingf/patterns/elementary/, 2001.

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

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

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

[14] D. Weintrop and U. Wilensky, "To block or not to block, that is the question: Students' perceptions of blocks-based programming," in *Proceedings of the 14th International Conference on Interaction Design and Children*, ser. IDC '15. New York, NY, USA: ACM, 2015, pp. 199–208. [Online]. Available: http://doi.acm.org/10.1145/2771839.2771860

[15] M. Dorling and D. White, "Scratch: A way to logo and python," in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '15. New York, NY, USA: ACM, 2015, pp. 191–196.

[16] 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

[17] C. Scaffidi and C. Chambers, "Skill progression demonstrated by users in the scratch animation environment," *International Journal of HumanComputer Interaction*, vol. 28, no. 6, pp. 383–398, 2012. [Online]. Available: https://doi.org/10.1080/10447318.2011.595621

[18] J. N. Matias, S. Dasgupta, and B. M. Hill, "Skill progression in scratch revisited," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, ser. CHI '16. New York, NY, USA: ACM, 2016, pp. 1486–1490. [Online]. Available: http://doi.acm.org/10.1145/2858036.2858349

[19] S. Dasgupta, W. Hale, A. Monroy-Hernández, and B. M. Hill, "Remixing as a pathway to computational thinking," in *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*, ser. CSCW '16. New York, NY, USA: ACM, 2016, pp. 1438–1449. [Online]. Available: http://doi.acm.org/10.1145/2818048.2819984

[20] B. M. Hill and A. Monroy-Hernández, "The cost of collaboration for code and art: Evidence from a remixing community," in *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, ser. CSCW '13. New York, NY, USA: ACM, 2013, pp. 1035–1046. [Online]. Available: http://doi.acm.org/10.1145/2441776.2441893

[21] K. Amanullah and T. Bell, "Evaluating the use of remixing in scratch projects based on repertoire, lines of code (loc), and elementary patterns," in *2019 IEEE Frontiers in Education Conference (FIE)*, in press.