Exploring the Programming Concepts Practiced by Scratch Users: an Analysis of Project Repositories

Ad Zeevaarders
Open University of The Netherlands
The Netherlands
adzeevaarders@gmail.com

Efthimia Aivaloglou
Open University of The Netherlands &
University of Leiden
The Netherlands
fenia.aivaloglou@ou.nl

***Abstract*—Scratch enables children to learn about programming by creating games and animations, and is currently one of the most popular introductory programming languages. While Scratch has been found to increase students' motivation and interest in programming, it has been debated whether Scratch users practice and learn about core programming concepts such as loops, conditional expressions, procedures and variables. This paper presents a large scale study of the progression of the programming concepts practiced by Scratch users through an analysis of their complete public project portfolios. A dataset of over 112 thousand authors and their 1 million projects was constructed and analyzed from three viewpoints. First, we investigate the development of programming concepts by looking at block usage statistics for each project in the users' repositories. Second, we score and analyze the dataset using a computational thinking rubric. Third, we identify users that have left the Scratch platform and evaluate the learning goals they have achieved. Our results show that, while users progress in Scratch, there is a positive trend in the use of all concepts that were examined. Within the least utilized concepts, even after the 20th project of Scratch users, are procedures, conditional loops and logic operations. Examining the users who have left the Scratch platform after creating at least the mean amount of nine projects, we measured that half had left without ever utilizing procedures, and a third had left without ever utilizing conditional loops.**

***Index Terms*—Programming education, Scratch, Game development, Computational Thinking, K-12**

I. Introduction

With computer science and computational thinking (CT) becoming increasingly mandatory in school curricula [1], there is a need to seek new ways of introducing programming concepts to young learners. One way in which this has been done is by the adoption of introductory programming languages, which serve as stepping stones towards more advanced programming languages by supporting users in learning basic programming concepts. Scratch is one of the most popular block-based languages of that kind, being used both in schools and through extra curricular activities [2]. The implementation of games and animations in the Scratch programming environment has been found to positively affect students' motivation and interest for programming [3].

Several studies have been carried out on the learning progression of users in block-based environments, which often explain progression by comparing performance differences in age groups or school grades [4]–[6]. In other cases, experimental courses or questionnaires are used to track short-term learning [7]–[11] or to measure existing knowledge in small groups of learners [12], [13]. Abstract measures of proficiency, such as vocabulary breadth and depth, which are not directly related to specific CT concepts, have also been used to infer learning progression [14]–[16] and have even sparked debate over the measured progression of Scratch users and the necessity of a large sample size [14], [16], [17]. Another topic of debate is the programming knowledge exhibited by the use of certain blocks, measured through their inverse document frequency [15], which was found to assign high weights to least-used blocks instead of blocks most indicative of programming knowledge [18].

The goal of this paper is to quantitatively analyze the progression in the use of core programming concepts by Scratch users, exploring it from the start of their learning trajectory to investigate which specific programming concepts are practiced throughout their experience on the platform. We are focusing on programming concepts that have been found to be hard for young learners, such as variables, procedures, conditional expressions and loops [4], [5], [8], [10], [12], [13], [19]. Because Scratch is designed to offer a first introduction to programming, we are also interested in exploring which concepts users have shown signs of practicing before leaving the Scratch platform, either to move to more advanced programming environments, or because of losing interest in programming. We aim to apply an automated approach for analyzing a large body of scraped project repositories in order to answer the following research questions:

RQ1 How do Scratch users progress in the use of elementary programming concepts such as variables, procedures, conditional expressions and loops?
RQ2 What is their progression on the application of CT skills, such as abstraction, data representation, flow control and logical thinking?
RQ3 Which programming concepts were practiced by users before they left the Scratch platform?

To answer our research questions, we scraped and analyzed the public Scratch project repositories of 112 thousand authors and statically analyzed the 1 million projects authored by them for the use of programming concepts and CT skills. The contributions of this paper are the following:

- an open-sourced set of software tools for scraping the

Scratch website for authors and their project repositories, including parsing logic for the two latest major Scratch versions,
- a public dataset of 1 million projects created by 112 thousand authors, parsed and labelled with the results of the automated analysis,[1] and
- an analysis of the repositories in the dataset in terms of progression in the use of programming concepts and application of CT skills.

II. Relevant Scratch Concepts

Scratch is a visual programming environment aimed at providing an easily approachable introductory programming experience to users. Scratch can be used as a stand-alone desktop client or as a browser client through the Scratch website. In Scratch, a project is called a *sketch* and programming syntax is represented by visual *blocks*. An example of a sketch is shown in Fig. 1. Running the sketch by clicking the green flag starts execution because of the top-most *event block* in Fig. 1; this block is a *hat* block, which serve as event hooks. Blocks come in a variety of *shapes* and each block belongs to a *category*. Block categories group blocks that are similar in functionality, such as *Looks*, *Motion* and *Operators*. Block connectors serve as visual guides showing which blocks can be composed in sequence. Some blocks can be embedded within other blocks. The *if/then* block in Fig. 1 has an *operator* block embedded into it as the condition to evaluate. This operator block itself has a variable embedded within it as part of the expression. A connected sequence of blocks, initiated by a hat block, forms a *script*. In Fig. 1, two scripts can be seen, one of which is a user-defined procedure called *askQuestion*. It spawns an input field and captures the entered string in a variable. Scratch uses *sprites* and *stages* as visual components. Sprites are akin to actors and have many visual properties, which can be manipulated to animate it, change its position or make it change size. Stages form the visual backdrop of the program. Scripts are defined within sprites and stages and run in that scope, but can communicate by broadcasting and receiving. All these features combined allow for complex behavior such as parallel execution of scripts or visual effects, showing that Scratch supports advanced programming constructs as well. The projects that users create are stored as packages either locally on disk or in the Scratch cloud. Project code and metadata is converted to JSON structures, while media, like sprites, accompanying the project are saved in separate files. The file format differs between versions of Scratch, with the most recent format being .sb3, coinciding with the release of Scratch 3.0, the current major Scratch version.

III. Related Work

Several works have statically analyzed Scratch projects for indications of learning of specific programming concepts. One of the first was that of Maloney et al., who analyzed 536

[1]The dataset and scraping software can be found at: https://github.com/ospani/zemitoolkit

Fig. 1. Example Scratch program highlighting several different syntax elements, including custom procedures.

Scratch projects for blocks that relate to various programming concepts, and found that within the least utilized ones are boolean operators and variables [20]. Tool support for the static analysis of Scratch programs has also been proposed; Dr. Scratch assigns one of three proficiency levels based on a rubric to CT concepts such as abstraction, logical thinking and data representation [21]. In [22], the use of programming concepts was examined in relation to the level of participation, the gender, and the account age of 5 thousand Scratch programmers. Xie and Abelson [18] analyzed the block usage, breadth and depth statistics of 10 thousand repositories containing over 20 projects and found that and among the least used computational concept blocks were lists, conditional loops, expressions and local variables. Aivaloglou and Hermans [19] analyzed 250 thousand Scratch projects scraped from the Scratch repository in terms of complexity, used programming concepts and code smells, and found that conditional loops and procedures were rarely used.

More related to our work on the learning progressions of Scratch users is the work by Scaffidi & Chambers [14] who analyzed sets of projects from user portfolios. Scraping a random sample of 250 users, their first projects and a random set of projects after the first, they analyzed them in terms of breadth (number of different block categories used), depth (number of blocks from a specific category used) and finesse, and found that the average breadth and depth decreases over time. A replication study by Matias et al. [16], using a full dataset of Scratch projects until 2012, challenged the results by Scaffidi & Chambers, showing that the average breadth and depth increased over time and attributing the results by Scaffidi et al. to poor sample size and data collection methods. These studies did not, however, evaluate the use of specific CT concepts. Lately, Amanullah and Bell [17] compared the first versus the last half of 35 thousand Scratch repositories, without however excluding remixed projects. The authors find

no progression in the usage of programming concepts such as loops and conditionals, and even encounter negative usage trends.

Related to learning progression, Seiter and Foreman [4] proposed a model for assessing the development of CT in Scratch users. The model assesses CT concepts related to procedures and algorithms, problem Decomposition, parallelization, abstraction and data representation, inferred from Scratch design patterns. The proposed rubric was used for manually analyzing 150 Scratch projects created by children from grades 1 to 6, finding that design patterns requiring understanding of parallelization, conditionals and, especially, variables were under-represented. Yang et al. [15] modelled learning progress as the vocabulary use of Scratch users over time. Using the dataset of Scratch projects until 2012, they generated vocabulary growth trajectories and clustered users according to their vocabulary usage in their first 50 projects projects and its growth. Infrequently used blocks were assigned higher weights as they were assumed to be indicative of advanced programming. De Souza et al. [10] investigated the evolution of CT using project source code that was generated during programming workshops. Rich et al. [23] studied the learning goals related to programming in order to develop three learning trajectories for the concepts of sequences, repetition and conditionals, which showcase the order and depth in which these concepts could be taught effectively.

IV. Methods

To answer our research questions, we created a scraper and parser tool, which we used to scrape the Scratch website for authors and their complete repositories of public projects. We then quantitatively analyzed the dataset for signs of improvement over time on programming concepts. Finally, we identified and analyzed the repositories of users that had left Scratch, to explore what they practiced during their stay.

A. Dataset

1) Scraping: To collect complete project repositories of users, the tool starts by scraping an initial set of random front-page authors from the Scratch API. Then, those authors' friends and followers are recursively scraped. Each author's data are then parsed into a relational database to form the set of authors whose repositories will be scraped. Next, the tool scrapes project metadata for each project, which includes the project name, view count, remix details and creation and modification dates. The project source code is then downloaded through the API and stored as JSON files on disk. To collect our dataset, the tool started scraping on the 1st of September 2019, until the 27th of October, 2019, and scraped 195,767 authors and 7,109,821 projects.

2) Parsing and Filtering: The tool parses projects by decomposing their JSON representation. Scratch uses three different formats (.sb1, .sb2 and .sb3) to save projects. These formats correspond to Scratch's major software versions. The sb2 and sb3 formats are both plain but different JSON structures, and constitute the majority of the projects on Scratch. To handle these different formats, separate parsing logic was written. The sb1 format is binary, and could therefore not be parsed.

The filters that we subsequently applied to the dataset were both on the scraped authors and on their projects. Authors that were found to have sb1 projects or projects that we had failed to parse, which amounted to 582,143 projects in total, were excluded from the dataset and from further analysis, since we need to keep only complete author repositories. For the remaining authors, the filter that we subsequently applied to their projects was that of empty projects and remixed projects, which were excluded from further analysis. This filter was applied because it is not possible to determine what an remixer's own contribution is relative to the original project, especially since original projects can evolve after they have been remixed, and version information is not provided by the Scratch platform. After excluding remixes, we further filtered out authors whose repositories consisted solely of remixes.

The filtering process resulted to the dataset that was used for the analysis, consisting of the repositories of 112,208 authors, containing a total of 1,019,310 self-created, non-empty projects. The source code of these projects was then parsed, resulting in 172 million blocks divided over 21 million scripts, which were stored in a relational database.

B. Concepts Evaluation

To find out which programming concepts were utilized and possibly improved upon by users for RQ1 and RQ3, we investigated the usage of procedure definitions, If & If/Else blocks, Repeat Until & Repeat Times blocks, Forever blocks and variables.

In order to arrive to quantitative measures of the CT skills demonstrated by the users for RQ2, we required an automated comprehension scoring model. There has been little consensus on how to best assess CT [24] because of the lack of consistent scoring criteria and the multitude of CT definitions available [25]. To select which model to apply, we specified the set of criteria a model should meet to be used in our study's context:

- assess the selected concepts,
- be automated or able to be automated,
- assign quantitative comprehension measures, and
- be compatible with Scratch blocks.

We reviewed relevant models [4], [7], [21], [26], [27] from the mapping study on CT comprehension models by Alves et al. [25] and chose the Dr. Scratch rubric [21] as the comprehension model, as it satisfied all of our criteria. The rubric maps CT skills of *Abstraction and Problem Decomposition*, *Parallelism*, *Logical thinking*, and *Data Representation* to four levels (None, Basic, Developing, Proficiency). We used the Dr. Scratch rubric to score our collected projects by translating its conditions for each of the proficiency levels to SQL queries and running them against our dataset of projects. For example, for a project to receive a 'Basic' score in Logical Thinking, it has to contain an if-block. The corresponding SQL query

TABLE I
TARGETED CONCEPTS PROJECTED ONTO THE DR. SCRATCH [21] SCORING RUBRIC.

Targeted concept	Dr. Scratch concept	Basic (Level 1)	Developing (Level 2)	Proficiency (Level 3)
Loops	Flow Control	Sequence of blocks	Repeat, forever	Repeat until
Expressions	Logical Thinking	If	If-else	Logic operations
Variables	Data Representation	Modifiers of sprite properties	Operations on variables	Operations on lists
Functions	Abstraction & Problem decomposition	More than one script and more than one sprite	Definition of blocks	Use of clones

checks if any blocks with the if-block opcode exist in the project. The mapping of our concepts to those used by Dr. Scratch, and the relevant conditions for each proficiency level, are shown in Table I.

C. Progression Analysis

To track the progression of users in the use of programming concepts, we ordered the projects within the user repositories by their Modified date. This makes it possible to follow progression from the first to the last project edited. We did not use the Created date to sort projects due to the possibility of authors coming back to older projects and editing them. Unlike studies that analyze the progression of learners in programming concepts by differentiating between grades or age groups (for example, [4]–[6]), we did not use these variables because they could not be obtained by scraping the Scratch website.

We then performed a statistical analysis, using a combination of SQL queries and R scripts to generate the information and visualizations necessary to answer our research questions. We further compared the first 3 with the last 3 projects of the authors who had at least 6 projects in their repositories to gain further insight in their progression.

D. Identifying Former Users

To identify users that have left the Scratch platform for RQ3, we calculated the difference in days between subsequent project modification dates for all the projects in all repositories we scraped. At the 95th percentile the difference in days is 41. We then took the latest project modification date in our dataset (27th of October, 2019) and subtracted 41 days from it. This date (16th of September, 2019) was used as the cutoff date for inactivity. An author that has no project with a modified date after that cutoff date is considered a f. In total, we analyzed 87,461 former author repositories containing 864,287 projects.

V. RESULTS

In total, we analyzed 112,208 unique author repositories containing 1,019,310 projects. The mean repository size in our sample was 9.08.

A. RQ1: Progression of Programming Concept Usage

For each of our targeted concepts, we plotted its use for the first 10 projects in the repositories, as any more would inhibit visualization. The results are shown in Fig. 3. For our analysis, repositories containing less than 10 projects were included as

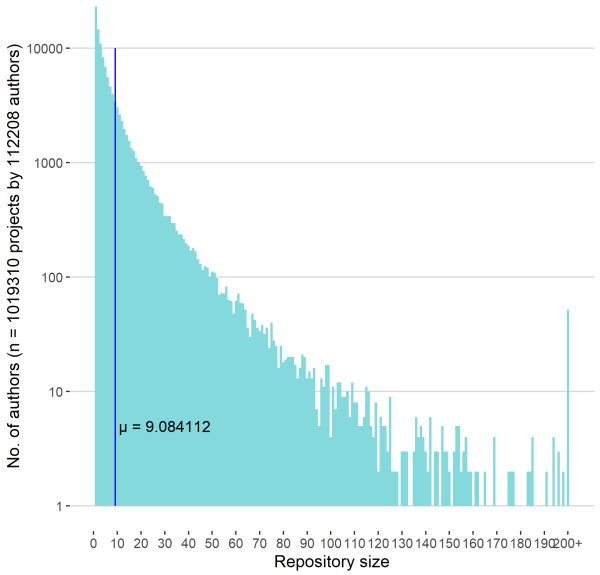


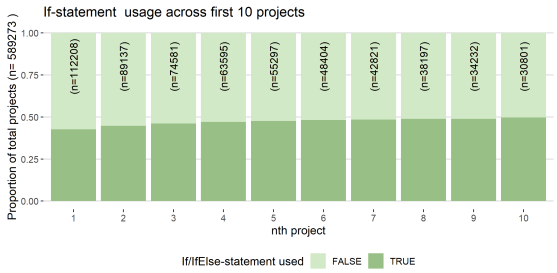
Fig. 2. Repository sizes in the dataset.

well. Fig. 4 plots, for the first 20 projects in the repositories, the number of users that have used each programming concept by their *n*-th project.

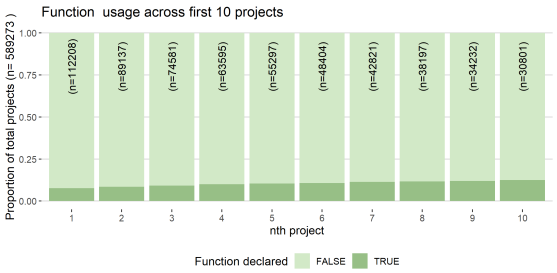
The distribution of *conditional expression* usage in the form of If and If/Else blocks is shown in Fig. 3a. These blocks were used in 42.6% of authors' first projects, and show an upward trend towards 49.7% at the 10th project. This proportion remains stagnant even at the 20th project, with 49.9% out of all 12,876 20th projects.

For *procedures*,we searched for projects that contained scripts that were procedure definitions. Of the first projects, only 7.58% contained procedures, though this increased to 12.5% at the 10th project, towards 14.5% at the 20th project. *Variable* usage was more frequent. User-created variables were used in 34.2% of authors' first projects and showed a slight upward trend from there, remaining stagnant at 43% around the 10th project and afterwards, being used in 44.4% of the 12,876 20th projects.

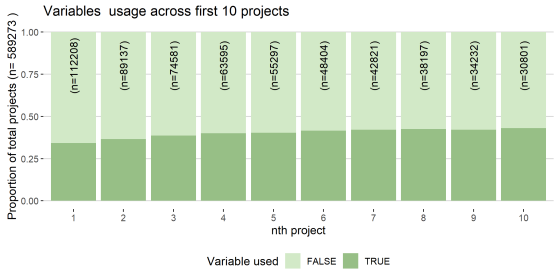
Regarding *loops*, *Forever* loops were encountered in 64.3% of first projects, while *Repeat Times* loops were more rare, existing in 35.6% of the first projects. *Repeat Until* loops, which require a conditional expression as a parameter, were encountered in just 14.5% of authors' first projects, again



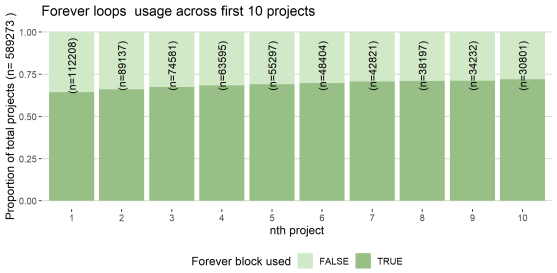
(a) Distribution of If/If-Else usage



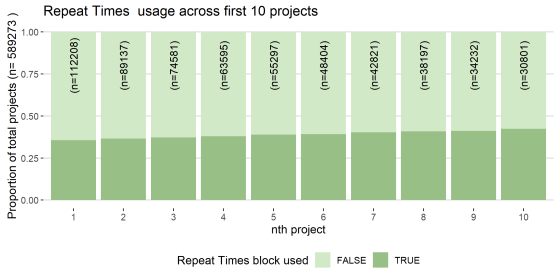
(b) Distribution of procedure declarations



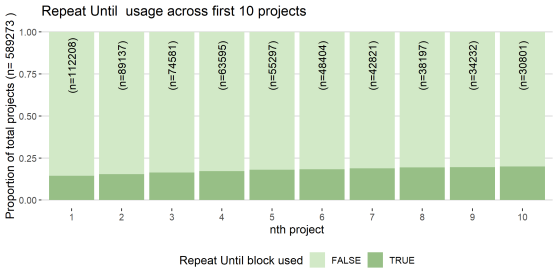
(c) Distribution of variable usage



(d) Distribution of Forever usage



(e) Distribution of Repeat Times usage



(f) Distribution of Repeat Until usage

Fig. 3. Distribution of programming concepts usage for the users' first ten projects.

showing an upward trend to 20% at the 10th project. After the 10th project, the progression slope becomes flatter with the proportion increasing to 22.6% at the 20th project.

B. RQ2: CT Rubric Evaluations

For the targeted CT skills we visualized the distribution of scores assigned by the Dr. Scratch rubric [21] in Fig. 5. Moreover, we compared the scores of the first three with those of the last thee projects of each author with six or more projects and visualized them in Fig. 6.

The concept of *flow control* (Fig. 5a) was utilized the most. 62.1% of users achieved a level of at least 2 in their first project, meaning they utilized Repeat Times and Repeat Forever blocks. An upward trend is visible for Level 3, which is the utilization of Repeat Until blocks. Level 1, which is the easiest score to attain as it requires at least one sequence of blocks, shows a downward trend.

For *logical thinking* (Fig. 5b), an upward trend is visible for Level 3, which is defined as use of logic operations. Here use of AND, OR and NOT blocks is seen to increase from 16.1% to 24.1%. For Level 2, which is the use of If/Else blocks, the proportions increase slightly, by 1.31% over 10 projects. Level 1, which is the use for If blocks, sees a slight decrease of 2.1%. The average logical thinking score of the first and last three projects of each repository is shown in Fig. 6, where higher scores are assigned to last projects.

For *data representation* (Fig. 5c), an upward trend for Level 3, which is the use of lists, is observed. Level 2, the use of variables, shows a slight upward trend as well. Level 1, the modification of sprite properties, decreases slightly, starting at 21.4% at the first project, towards 20.8% at the 10thproject.

Last, the *abstraction* concept (Fig. 5d) shows an upward trend for Level 3, which is the use of clones. For Level 2, the definition of procedures, an upward trend is observed as well. Level 1, which requires a project to have more than one script and sprite, shows a slight downward trend due to adoption of higher levels, as the proportion of users who did not utilize any abstraction concepts remained stagnant, even up until the 20th project. In Fig. 6, a proportional decrease in average scores up until Level 1 is observed.

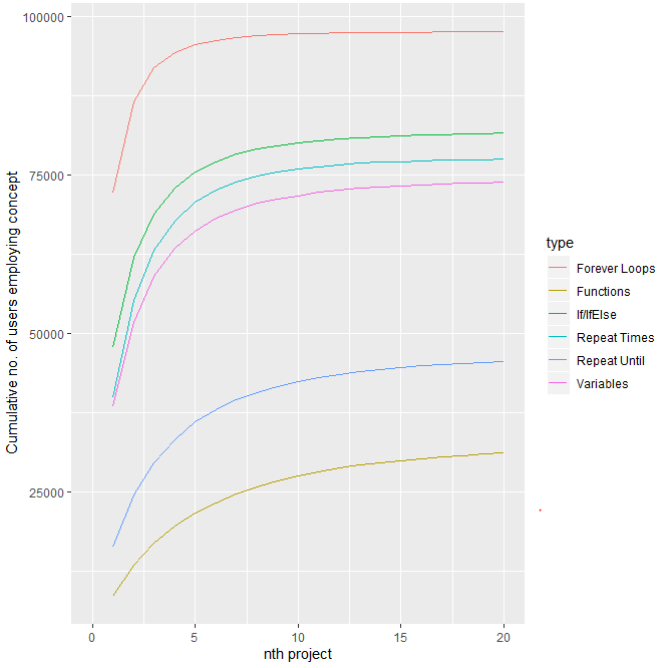


Fig. 4. Number of users having used a programming concept by their n-th project (total users: 112,208).

TABLE II
NUMBER OF USERS THAT HAVE LEFT THE SCRATCH PLATFORM WITHOUT HAVING USED A PROGRAMMING CONCEPT, OUT OF THE 87,461 FORMER USERS AND THE 29,958 COMPLETE FORMER USERS.

Programming concept	**Former users not using concept**	**Complete former users not using concept**
If/If-Else	20,780 (23.76%)	2,530 (8.44%)
Procedures	59,858 (68.44%)	13,838 (46.19%)
Variables	26.538 (30.34%)	3,555 (11.87%)
Forever Loops	8,392 (9.59%)	307 (1.02%)
Repeat Times	22,254 (25.75%)	1,698 (5.67%)
Repeat Until	48,263 (55.18%)	9,982 (33.32%)

C. RQ3: Concepts Practiced Before Leaving the Scratch Environment

For RQ3 we analyzed the repositories of the 87,461 users that were identified as inactive, which we refer to as former users. We separately analyzed the set of former users with a repository size equal to or above the sample mean of 9 projects (See Fig. 2), which is 29,958 authors with 647,249 projects. We call these complete former students in our analysis below.

Table II presents the number of users that left Scratch without having used each of the programming concepts that we explored. Especially when considering the complete former users, only a small percentage left Scratch without ever utilizing conditional statements (8.44% out of 29,958 complete former users), variables (11.87%), Forever loops (1.02%), and Repeat Times loops (5.67%).

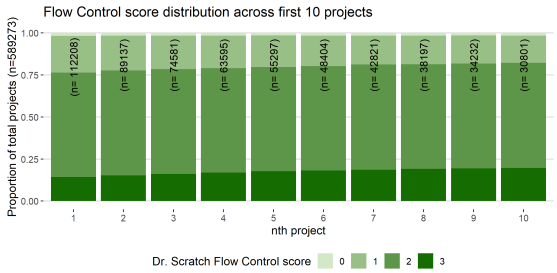
Within the most under-utilized concepts from former users are procedures and conditional loops. Regarding procedures, the amount of former users that never defined a procedure is 59,858, or 68.44% of all former users. Examining the complete former users, we find that 13,838 had no procedure definitions, meaning that 53.81% of users that created at least 9 projects defined a procedure at least once during their stay on the Scratch platform. Examining *repeat until* loops, the amount of complete former students that never used them is 9,982 or 33.32%.

VI. DISCUSSION

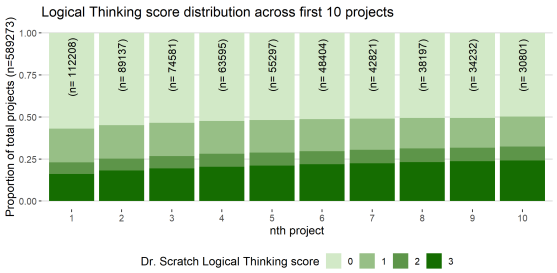
Our findings indicate that Scratch users progress in the use of several computational thinking concepts like abstraction & problem decomposition, flow control, logical thinking and data representation as they create more projects. This is not in agreement with the findings in [17], where limited or negative progression was found in the usage of programming concepts such as loops and conditionals. We believe that this difference can be attributed to the larger sample size that we used, the exclusion of remixed projects, and the fact that we analyzed the data from various viewpoints.

Specifically, the analysis of the user's progression was made from three viewpoints: that of elementary programming concepts, using a rubric that evaluates CT skills, and through the concepts practiced by former users. Working with this approach, the benefits that we find are twofold. First, it captures that not all programming concepts should or need to be employed in every project to convey that a user is advanced. Focusing solely on the progression as evaluated by the use of programming concepts or the rubric can be misleading, because it does not give a full image of the learning progression. For example, examining the use of variables throughout the first 20 projects of users we found a positive trend from 34.2% in the first projects to 44.4% of the 20th projects, which is also in line with the findings of Aivaloglou and Hermans, where variable usage was found in 31.51% of Scratch projects [19]. This view alone would be misleading, because it could be interpreted as an under-utilization of variables from users even after creating 20 projects. However, when examined from the last viewpoint, that of the concepts practiced by former users, we find only 30% having never used them and, more importantly, that the majority (88.13%) of the users who have left Scratch after 'seriously' using it (creating at least the mean amount of nine projects) have used variables at least once, indicating that users do practice the concept of variables while using Scratch.

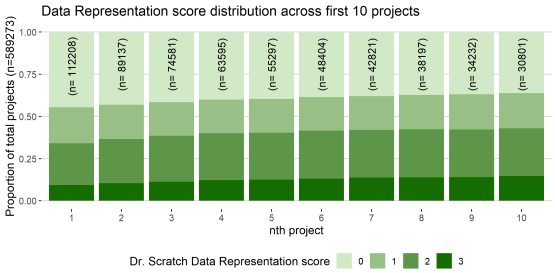
The second benefit from adopting this approach is that it enabled us to better capture progression and the lack of it. For example, the under-utilization of logic operations was not brought up when examining the elementary concepts, since logic operations can be used in conditional expressions, loops, and other expressions, but as an element in the logical thinking dimension of the CT skills rubric. It is also observed here that the high If/If-Else block usage contrasts with the low conditional loop usage, since both require a conditional expression. This could be attributed to a lack of understanding of how the conditional expression operates with loops.



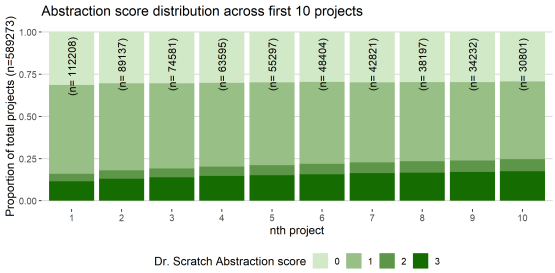
(a) Distribution of flow control scores



(b) Distribution of logical thinking scores



(c) Distribution of data representation scores



(d) Distribution of abstraction & problem decomposition scores

Fig. 5. Distribution of CT skills evaluation.

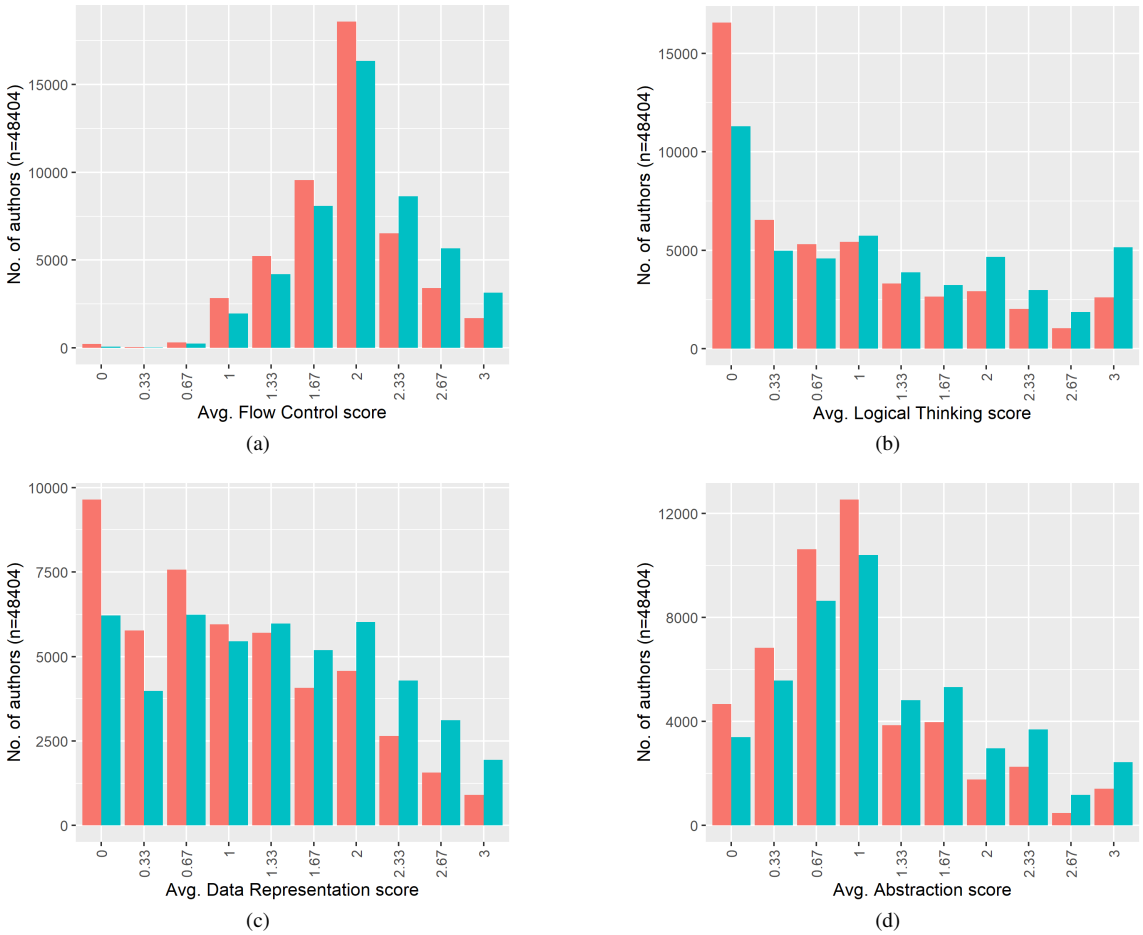


Fig. 6. Average CT concept scores of first three (left/red bars) and last three (right/blue bars) projects in author repositories.

The under-utilization of procedures was evident across all viewpoints, with their use remaining as low as in 14.5% of 20th projects, with almost half of the users who have left the Scratch platform after creating at least nine projects never having created a procedure. Low procedure usage was also reported in existing work in the Scratch repository [19], where only 7.7% of 233,491 projects were found to utilize them, as well as by Troiano et al. [11], who observed little progression in abstraction concepts beyond Level 1 in their analysis of the Dr. Scratch scores of 317 projects created in 8th grade, where only 18 projects used procedures. Our analysis confirms those findings in a large set of users and their project portfolios. This very essential programming concept is therefore rarely practiced, which can be attributed both to limitations imposed by the Scratch environment, like the local scope of procedures, and to the difficulty for internalizing certain CT concepts before a certain age [4].

VII. Limitations

The dataset that was constructed and used for the analysis contains all projects that the scraped users had made public, but not their private projects, which cannot be obtained. In our analysis, this might have influenced the repository size and former user calculation, as well as the ordering of the projects, as public projects alone do not capture the full extent of an author's activity. The exclusion of remixes might have influenced the same aspects. However, including remixes would have skewed our analysis results, since remixed projects should not be considered authored projects and it is not possible to distinguish with the available information a remixer's own contribution relative to the original project.

Regarding the ordering of projects, we used the latest modification date of projects to order them chronologically. Another option would be use the project creation dates. However, since users can go back and edit their projects at any time, that might not capture the true chronology of a user's activity. Regarding our sample of authors, we might have captured only a local sample of Scratch authors due to our scraping method: by scraping friends and followers, and their friends and followers and so on, we might collected only authors that have some relationship towards each other.

It should be recognized that a limitation to this study is that, being based solely on quantitative data acquired from the Scratch project repository, the results concern the application of programming concepts through the use of Scratch blocks. Even though these indicate that the project authors practiced applying specific programming concepts, our results cannot be extended to proving the acquisition of CT skills or the authors' efficacy or comprehension of programming concepts, which would require qualitative insights about the programs and the process that they were following.

VIII. Conclusion

In this paper we presented a quantitative study on a large body of scraped Scratch repositories. By analyzing the projects of these repositories for use of loops, expression, variable and procedure concepts, both individually and using a rubric, we explored their progression. Across the board, the results show an increase in concept utilization and CT scores for active and former users alike. We observe that logic operations, conditional loops and procedures are used very little, with two-thirds of the users that left the Scratch environment doing so without ever using procedures in their public repositories. To support the replicability of our findings, we further contribute the scraping software and the entire dataset, from the raw scraped files to the parsed relational data and all scripts and analysis files in between.

Several directions for future work can be supported by the available dataset, especially because it contains entire user repositories instead of random projects. Further research can focus on the learning progressions of Scratch users using variables not considered in this study, such as code quality metrics and code smells, to highlight other aspects of learning. Learning trajectories can also be investigated, to explore the order by which concepts are attained by learners, which is made possible by the chronology of the projects in the dataset. The under-utilization of logic operations, conditional loops and procedures, even after creating several Scratch projects and even leaving the Scratch environment, is a topic that could be researched further, because it is not known if it is the result of the Scratch environment design or of learning difficulties related to these concepts.

References

[1] K. Falkner, S. Sentance, R. Vivian, S. Barksdale, L. Busuttil, E. Cole, C. Liebe, F. Maiorana, M. M. McGill, and K. Quille, "An international comparison of k-12 computer science education intended and enacted curricula," in *Proceedings of the 19th Koli Calling International Conference on Computing Education Research*, ser. Koli Calling '19. ACM, 2019.

[2] E. Aivaloglou and F. Hermans, "How is programming taught in code clubs? exploring the experiences and gender perceptions of code club teachers," in *Proceedings of the 19th Koli Calling International Conference on Computing Education Research*, ser. Koli Calling '19. ACM, 2019.

[3] I. Ouahbi, F. Kaddari, H. Darhmaoui, A. Elachqar, and S. Lahmine, "Learning basic programming concepts by creating games with scratch programming environment," *Procedia - Social and Behavioral Sciences*, vol. 191, pp. 1479 – 1482, 2015, the Proceedings of 6th World Conference on educational Sciences. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877042815024842

[4] L. Seiter and B. Foreman, "Modeling the learning progressions of computational thinking of primary grade studentsf," in *Proceedings of the ninth annual international ACM conference on International computing education research*. ACM, 2013, pp. 59–66.

[5] F. Hermans and E. Aivaloglou, "Teaching software engineering principles to K-12 students: a MOOC on Scratch," in *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET)*. IEEE, 2017, pp. 13–22.

[6] I. N. Šerbec, Š. Cerar, and A. Žerovnik, "Developing computational thinking through games in scratch," *Education and Research in the Information Society*, pp. 21–30, 2018.

[7] A. Funke, K. Geldreich, and P. Hubwieser, "Analysis of scratch projects of an introductory programming course for primary school students," in *2017 IEEE global engineering education conference (EDUCON)*. IEEE, 2017, pp. 1229–1236.

[8] M. Mladenovic, I. Boljat, and Z. Zanko, "Comparing loops misconceptions in block-based and text-based programming languages at the k-12 level," *Education and Information Technologies*, vol. 23, pp. 1483–1500, 07 2018.

[9] E. Förster, K. Förster, and T. Löwe, "Teaching programming skills in primary school mathematics classes: An evaluation using game programming," in *2018 IEEE Global Engineering Education Conference (EDUCON)*, 2018, pp. 1504–1513.

[10] A. A. de Souza, T. S. Barcelos, R. Munoz, R. Villarroel, and L. A. Silva, "Data mining framework to analyze the evolution of computational thinking skills in game building workshops," *IEEE Access*, 2019.

[11] G. M. Troiano, S. Snodgrass, E. Argımak, G. Robles, G. Smith, M. Cassidy, E. Tucker-Raymond, G. Puttick, and C. Harteveld, "Is my game ok dr. scratch? exploring programming and computational thinking development via metrics in student-designed serious games for stem," in *Proceedings of the 18th ACM International Conference on Interaction Design and Children*, 2019, pp. 208–219.

[12] A. Swidan, F. Hermans, and M. Smit, "Programming misconceptions for school students," in *Proceedings of the 2018 ACM Conference on International Computing Education Research*. ACM, 2018, pp. 151–159.

[13] S. Grover and S. Basu, "Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and boolean logic," in *Proceedings of the 2017 ACM SIGCSE technical symposium on computer science education*. ACM, 2017, pp. 267–272.

[14] C. Scaffidi and C. Chambers, "Skill progression demonstrated by users in the scratch animation environment," *International Journal of Human-Computer Interaction*, vol. 28, no. 6, pp. 383–398, 2012.

[15] S. Yang, C. Domeniconi, M. Revelle, M. Sweeney, B. U. Gelman, C. Beckley, and A. Johri, "Uncovering trajectories of informal learning in large online communities of creators," in *Proceedings of the Second (2015) ACM Conference on Learning@ Scale*. ACM, 2015, pp. 131–140.

[16] 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*. ACM, 2016, pp. 1486–1490.

[17] K. Amanullah and T. Bell, "Analysis of progression of scratch users based on their use of elementary patterns," in *2019 14th International Conference on Computer Science & Education (ICCSE)*. IEEE, 2019, pp. 573–578.

[18] B. Xie and H. Abelson, "Skill progression in mit app inventor," in *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2016, pp. 213–217.

[19] 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*. ACM, 2016, pp. 53–61.

[20] J. H. Maloney, K. Peppler, Y. Kafai, M. Resnick, and N. Rusk, "Programming by choice: Urban youth learning programming with scratch," in *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*. ACM, 2008, pp. 367–371.

[21] J. Moreno-León, G. Robles *et al.*, "Dr. scratch: a web tool to automatically evaluate scratch projects." in *WiPSCE*, 2015, pp. 132–133.

[22] D. A. Fields, M. Giang, and Y. Kafai, "Programming in the wild: Trends in youth computational participation in the online scratch community," in *Proceedings of the 9th Workshop in Primary and Secondary Computing Education*. ACM, 2014, pp. 2–11.

[23] K. M. Rich, C. Strickland, T. A. Binkowski, C. Moran, and D. Franklin, "K-8 learning trajectories derived from research literature: Sequence, repetition, conditionals," in *Proceedings of the 2017 ACM conference on international computing education research*, 2017, pp. 182–190.

[24] S. Grover and R. Pea, "Computational thinking in k–12 a review of the state of the field," *Educational Researcher*, vol. 42, pp. 38–43, 02 2013.

[25] N. D. C. Alves, C. G. Von Wangenheim, and J. C. Hauck, "Approaches to assess computational thinking competences based on code analysis in k-12 education: A systematic mapping study," *Informatics in Education*, vol. 18, no. 1, p. 17, 2019.

[26] B. Boe, C. Hill, M. Len, G. Dreschler, P. Conrad, and D. Franklin, "Hairball: Lint-inspired static analysis of scratch projects," in *Proceeding of the 44th ACM technical symposium on Computer science education*. ACM, 2013, pp. 215–220.

[27] C. G. Von Wangenheim, J. C. Hauck, M. F. Demetrio, R. Pelle, N. da Cruz Alves, H. Barbosa, and L. F. Azevedo, "Codemaster–automatic assessment and grading of app inventor and snap! programs." *Informatics in Education*, vol. 17, no. 1, pp. 117–150, 2018.