Evaluating the Use of Remixing in Scratch Projects Based on Repertoire, Lines of Code (LOC), and 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*—This Full Paper in the Research Category evaluates the use of remixing in Scratch. A feature of the Scratch programming environment is that it supports students to share their code and "remix" (modify) other students' code. Remixing in Scratch has garnered much attention by the research community as use of collaboration for learning was one of the main ideas behind Scratch. It can provide opportunities to read others' code, learn how features can be implemented using the Scratch language, and contribute to the program. It can also prevent students from engaging with the code if they copy an existing program that does what they are trying to do without needing modification. The literature shows mixed results regarding use of remixes in Scratch. We have investigated at a large scale what happens in practice by analysing thousands of student programs shared through the Scratch online repository. As well as replicating prior work on a larger scale to show the impact of remixing on learning programming skills through Lines of Code (LOC) and repertoire of block usage, we also measure the use of elementary patterns (common combinations of commands). We track the progress of each project through its remixes and compare the results between the root version and the final version.**

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

I. Introduction

Scratch was designed and created with collaboration as its one of the main features [1]. It has enjoyed considerable success, and has given millions of students experience with programming. More than 37 million projects have been shared in the online community, and over 35 million users have been registered on the Scratch website since it was launched, and these numbers are still very much growing. In this research we examine the programming skills that students develop, particularly their repertoire and use of elementary patterns, taking account of how those skills are exercised in the context of programs built using "remixes" of other students' work.

"Remixing" is a key form of collaboration in Scratch, where a student takes a copy of another student's program, and modifies it in their own space, which can then be further shared with others for remixing. The literature offers two sides of the same picture when it comes to remixing of work. Working with peers has been linked to higher quality results, and there is evidence in the literature to support this argument [2], [3], including when learning computer programming [4]. On the other hand, there is some evidence that increased collaboration doesn't necessarily bring out the best results, and in fact can lead to poorer products [5], [6], and any form of peer instruction needs to be structured well to succeed. In this study we offer analysis of what is happening in publicly shared projects, especially those involving remixing, to explore what remixes do and don't achieve in Scratch.

Remixing in Scratch has been the recipient of much attention and the idea of peer supported learning clearly has merit, but results have been mixed so far. Remixing has been shown to promote learning [7] but there is also a danger that if there is poor coding style or hidden bugs in a project, these might also propagate to remixed projects and contribute in learning bad programming practices, contrary to the goal of remixing [8]. In this study we build upon the results from prior studies on remixing in Scratch, and attempt to present a clearer picture through measuring the use of elementary patterns in remixes. While Scratch may not be intended to teach more structured programming style such as controlled loops, it has all the features needed, and has been used in context where such skills are expected to the learnt [9], which raises the question of whether examples of these patterns appear in the examples that students are remixing, and if they are learning from them.

The elementary patterns [10] chosen for this study (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.

Previous studies have tried to measure the impact of remixes on learning and skill progression mainly on the basis of occurrence of atomic programming constructs belonging to different categories. These studies didn't measure the sophistication of programs in depth. When we talk about remixing leading to learning and skill progression there must be enough evidence

in the data to suggest the same, and while learning more blocks in Scratch is an achievement, learning structures for putting them together is also an important element of programming — learning and skill progression should lead to more sophisticated use of programming and programming elements. We address this by adding an extra layer of sophistication to previous studies by using elementary patterns to analyze programs.

TABLE I
PATTERNS USED IN ANALYSIS

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

We analyzed around 1.5 million Scratch projects and only selected projects that were remixes (not original), and then worked our way back to the root of series of remixes to investigate how many levels of remix each project had, and the effect of levels of remixes on the project, particularly how the original and remixed projects compared with respect to lines of codes (LOC, which in Scratch is the number of blocks in a program) and use of elementary patterns. Use of elementary patterns and the resulting change (if any) between original and remixed will indicate how much progress a user makes based on the use of patterns. An important thing to consider is the increase in the LOC might not necessarily show an increase in programming skills, and in fact the opposite might be true as better programming skills can lead to using fewer LOC, with better use of sophisticated programming constructs. Therefore, using elementary patterns is more compelling for measuring sophistication, since an increase in LOC may simply reflect bad practice, such as copying and pasting a sequence of instructions over and over.

The analysis below is divided in two parts; in the first part we re-test the hypotheses from a previous study based on *repertoire* and the use of CT concepts for all Scratch projects shared by a sample of users, comparing remixed and original projects. This analysis is covered in section III. In the second part we measure the progression in skills through students' use of elementary patterns, by tracking the changes from the root project to the final version of the project; this is covered in section IV.

For the sake of accuracy, in section V we also consider the difference in the total number of blocks between each remix in the second part of the analysis for projects having a remix depth of greater than 1. This is because there could be any number of remixes between the root and remixed projects, and one of the remixes might have totally changed the appearance from the root project.

II. Previous Work

The study in [14] tries to measure the progression of Scratch users over time. They make use of four models to come up with the results. The first one is Onion model [15], [16] used to measure social skills of Scratch users. The second one was an adapted version of a competence model [17], [18] to compute breadth, depth, and finesse. Breadth and depth was measured by counting the distinct categories and the number of primitives within each category respectively. Finesse was measured by counting the type of problems solved but it was essentially measuring similar primitives to breadth and depth. The third model derived from their own work; [19], [20] tries to measure the abstraction but end up measuring all the primitives used in Scratch due to the absence of macros and functions in Scratch. The fourth and the last model adapted the COCOMO model [21] to measure progression based on the saved history of projects. We use elements from each of these models (not in exactly the same form) for the purpose of this study. The results of this study [14] showed a negative trend for technical skill progression, although Matias et al. [22] revisited the work done by Scaffidi and Chambers [14] and obtained opposite results to some of their findings.

III. Repertoire Analysis

Dasgupta et al. [7] have shown that remixing can be beneficial in learning computational thinking concepts. Their analysis is based on two hypotheses that largely measure the growth in the number of programming block types ("repertoire"), and the use of particular programming concepts.

To replicate these results, we analyzed all projects shared by 9,141 users, selected randomly from the list of users used in [23]. To match the work we were replicating, we kept the standard blocks provided by Scratch for analysis and discarded any custom blocks (even though they might represent a higher form of sophistication). We used the Scratch API to get the code of all Scratch projects by each user in JSON format, and then used text parsing in Python to look for the presence of each block type. As in the work being replicated [7], we measured the *cumulative repertoire* of a user, which is the total number of distinct blocks that they showed in all of

their shared projects to date. Table II shows the cumulative repertoire observed.

TABLE II
COMPARISON WITH PREVIOUS RESULTS

Current	M: 34	mean: 35	std: 20	R: [0, 131]
Past [7]	M: 23	mean: 28	std: 21	R: [0, 142]

These differ somewhat from the results being replicated [7] — the median and mean values are higher in our results, whereas the standard deviation is similar. The difference could be due to many factors. Scratch version 1.4 was in use until 2012, so earlier work may have a higher proportion of version 1.4 programs. A number of blocks that were available in Scratch 1.4 have either been depreciated or merged into other blocks. Most of the projects analyzed by us are after 2012, and created with Scratch 2.0. Also, the popularity of Scratch has increased enormously over the years and it is quite possible that people have been using and sharing projects in Scratch more frequently than was the case in 2012. In Scratch 1.4 there were 128 blocks in total excluding variations, while in current version of Scratch there are 119 blocks. How variations of different blocks was analyzed could also be a cause of the difference, and our analysis is based on users who have at least one non-empty project (so the minimum repertoire is 1 as compared to 0 in the study being replicated [7]).

To see the impact of remixing on the block repertoire we plot (Figure 1) the number of remixes against the cumulative repertoire (this addresses hypothesis H1 in the prior study, which was that *changes in a users programming repertoire will be larger when she has engaged in more remixing activity*). We didn't calculate the control variables, as the purpose in this case is to look at the activity of remixing. Out of 9,141 users selected for analysis, 57% had engaged in remixing.

The graph suggests that number of remixes have no substantial impact on repertoire except in extreme cases, or least the relationship is not very clear. To get a better picture we calculate averages and compare the graphs of users who had no remix vs. the users who had at least one remixed against the cumulative repertoire.

Out of 9,141 users, 3,884 had no projects that were remixes of any other project, while 5,257 users had one or more remixed projects. The comparison of users who had remixed projects versus the users with no remixes is shown in Table III, which shows that there is a much larger cumulative repertoire for users who had remixed one or more projects. The average cumulative repertoire for the users with one or more remixes is 41, whereas the users with no remixes score 27 on the same measure. This evidence could suggest that remixing is promoting learning (the use of more programming blocks) but it could

TABLE III
REPERTOIRE FOR REMIX VS. NO REMIX

remix	M: 41	mean: 41	std: 21	R: [1, 131]
no remix	M: 25	mean: 27	std: 16	R: [1, 104]

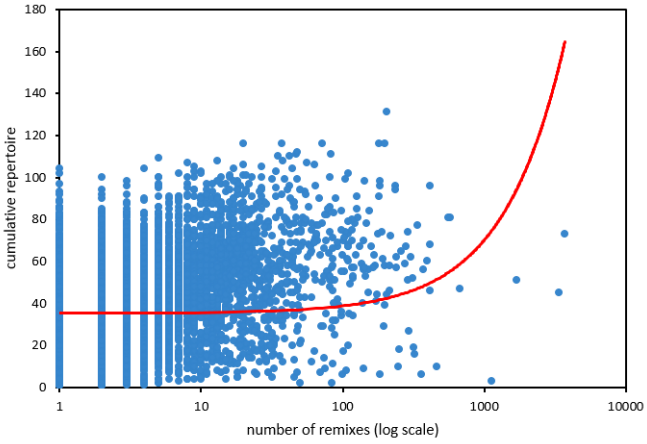

Fig. 1. Repertoire for remix vs. no remix; regression line shown in red

also mean that users are just copying other people's code and using it without making any changes or understanding it. To investigate this, in Table IV we show the average cumulative repertoire for all users involved in remixing activity, but only in their original projects and excluding their remixed projects, compared with the averages of users with no remixing.

TABLE IV
REPERTOIRE FOR REMIX VS. NO REMIX (ORIGINAL FILES OF REMIXERS ONLY)

remix	M: 25	mean: 27	std: 22	R: [1, 131]
no remix	M: 25	mean: 27	std: 16	R: [1, 104]

This reveals an interesting result. There is no difference in repertoire between the "no remix" students, and the repertoire of the original programs written by the students who worked with remixes. It appears that remixing is being used to copy code without learning new concepts or understanding the code.

The fitted regression line (red line) also shows upward trend for cumulative repertoire with the increase in number of remixes.

To test whether more advanced learning can take place, we now use elementary patterns (Table I) for the analysis instead of Brennan and Resnick's mapping of Scratch blocks to CT concepts [24] that were used previously in [7]. We measure the occurrence of important programming concepts and elementary patterns in both original and remixed projects for each user, and show the results of different combinations in Table V.

There were 5,257 users out of 9,141 (57.51%) who had one or more remixed projects and the remaining 3,884 (42.49%) users had no remixed project. Hypothesis H2 (from [7]) suggests that users with more remixes are likely to learn new CT concepts and then use these new concepts in their original (non remixed) projects. In Table V we try to show a broader view of the overall situation. We compare different combinations to test the role of remixing in learning new CT concepts. The aim is to reveal the impact of remixing activity on the use of CT concepts.

TABLE V
ELEMENTARY PATTERNS IN BOTH ORIGINALS AND REMIXED

Item	Users with one or more remix (5257)			Users with no remix (3884)
	Used in both originals and remixed (%)	Used in originals only (%)	Used in remixed only (%)	(%)
if	52.62	12.14	18.20	70.21
ifelse	20.92	10.98	28.61	25.44
nestedif	15.16	11.19	21.72	17.71
list	4.20	4.30	18.77	3.76
repeat	48.60	15.00	24.28	63.73
forever	66.96	9.17	16.72	79.27
until	18.09	9.28	31.16	23.97
search	10.63	15.08	13.81	24.23
pal	0.99	2.05	13.14	1.47
ls	0.00	0.08	0.32	0.03
gls	0.00	0.06	0.13	0.00
laah	0.15	1.85	1.86	1.03
pl	0.53	2.61	6.52	3.35
var	46.28	13.87	23.15	62.31

There are a few observations that can be made from the results of the table. We see that it cannot be guaranteed that a CT concept used in remixed projects will be learned or adapted by users remixing the projects, although the likelihood of using *some* blocks increases. The frequent blocks with higher usage percentages (if, repeat, forever) are likely to be used anyway, no matter the amount of remixing. On the other hand, the usage of more sophisticated blocks and patterns (list, pal, ls, gls, laah, pl) does not seem to be influenced by remixing, and their usage remains very low. For example there were 31.16% of the users who had used an "until" block in remixed projects, but it wasn't found in any of their own original projects. A similar case can also be seen with "Process All Items" (pal), where 13.14% of users had this in a remix, but only 0.99% used in their own original programs when they had observed it in a remixed program. This provides further evidence to the observation made earlier that users might just be copying the code and not learning from it. Of course, remixing may be playing other important roles, such as enabling students to build confidence or gain experience with larger projects, and just because a student doesn't use something they have observed in original projects doesn't mean that they didn't engage with it in the remixed project. However, while basic constructs like "forever" are being picked up, patterns involving more than one construct don't appear to be.

IV. LOC AND USE OF PATTERNS DURING REMIXES

In order to investigate this more carefully, we investigated how LOC and use of patterns changes with each remix of a project. To do this, we took 1 million Scratch projects randomly selected from a list of projects ids (120000000 – 200000000). The random selection helps to remove some of the threats to validity (such as only focusing a smaller point in time, or projects from a specific user) and provides a large enough sample to be representative of the population. Out of 1 million processed projects only 150,320 were still accessible — other projects were were either deleted or not shared anymore. We also took two further samples and processed 296,962 and 210,250 Scratch projects respectively by requesting project IDs in the sequence (260895723 – 260598761) and (228653285 – 228863535). Using three samples addresses the possibility of using a non representative sample (contiguous projects are more likely to cover a related group of students or projects written at a similar time). For the first sample, out of 150,320 projects, 41,734 were remixed and rest of the 108,586 projects were original. This gives us a percentage of 27.76, which is fairly representative of the population, as in the analysis of all three samples the percentages were similar (29.75, 27.62, 27.76). Since the difference is minor so we combined the results of all of the samples.

In the first step we selected only the projects that were remixes of other projects i.e. we discarded all the projects that were original projects (not remixes of any other projects). This gave us 64,372 projects out of 1.5 million projects. This low proportion is due to many projects that were not shared anymore, or deleted, combined with our criteria for the program being a remix.

One of the key reasons for analyzing remixed projects is to investigate whether remixing promotes progression in programming skills. One way to check this is to count the number of elements (line of codes or blocks, referred to as LOC) added with each remix starting from the root project (a remixed project that isn't a remix itself). We found the average number of blocks in remixed projects and compared it with average number of blocks in root projects, which was 283.58 and 253.54 respectively. That is, there is an average difference of 30 blocks added to root projects, ignoring the level of remixes (remix depth) between the root and the remixed project. Some of the interesting results are shown in Table VI.

TABLE VI
REMIX STATISTICS

Average number of blocks (LOC) in root projects	348.46
Average number of blocks (LOC) in remixed projects	376.82
Percent of projects with no change from root project	20.92%
Percent of projects with no blocks	2.88%

We show the five number summaries for root projects compared with remix projects in Table VII. This shows a definite increase in the number of blocks in remixed projects compared to root projects. Of course, this raises the question of whether this increase a sign of better programming skills or not. Also, if a project has multiple remixes from root to final version, it could be remixed by users of varying skills. Based on the results from previous sections, an increase in the number blocks is not a clear sign of learning, although it does show that most of the time users are embellishing projects, although about one in five projects are simply copied from the parent project.

TABLE VII
FIVE NUMBER SUMMARY FOR ROOT AND REMIX BLOCKS

	min	Q1	median	Q3	max
Root	0	2	22	92	49418
Remix	0	14	39	131	49418

We track the progress of each project through remixes and show the difference between the root project and the final version. This will show us how much contribution remixing has made to projects. We have also kept our analysis limited to blocks which are closest to computational thinking concepts, in our opinion.

To obtain a more meaningful measure, we return to the LOC, especially of simple patterns, as these can suggest more sophisticated use of programming elements. Table VIII shows the occurrence of important programming elements in Scratch programs for root and remixed projects. The second column, "Root", shows the average number of particular blocks in root projects, and the "Remix" column shows the average number of blocks in remixed projects. The "Usage" columns are there to help develop a better understanding of the difference, as they show the percentage of projects having the specific block in the root and remixed projects respectively.

TABLE VIII
REMIX AND LEARNING

Item	Root	Remix	usage root	usage remix
if	7.24	8.02	35.67	44.34
ifelse	3.31	3.57	19.22	24.52
nestedif	12.73	12.87	15.68	18.11
list	1.70	1.68	8.07	9.19
repeat	3.94	4.47	37.57	49.24
forever	4.52	5.74	52.43	67.13
until	1.64	1.69	21.32	23.95
variable	8.91	9.32	36.44	41.93

All these numbers enable us to view a better picture of the actual situation and help us to interpret the data. We see a slight increase in the average number of each block in remixed projects, except for "list", where it is decreases slightly, but this increase is minor. The usage of each block does increase notably in remixed projects for most blocks though, and it can be stated that remixing does improve the repertoire of blocks to some extent. Another important point to consider is that although usage is increasing, still the usage of important blocks is quite low (less than 50% except for "forever"). This highlights that more than 50% of the projects are missing the use of basic programming elements.

We also checked for some higher forms of programming skill in the shape of elementary patterns in remixed projects, compared with their root projects. Table VIII showed that although the coverage for important programming elements was 50% and below, they could still be found consistently. Now we show what happens when we add an extra layer of sophistication when we check for the use of elementary patterns (loop-based patterns). The results are shown in Table IX.

TABLE IX
REMIXING AND LEARNING: ELEMENTARY PATTERNS

Item	Root	Remix	usage root	usage remix
search	0.29	0.31	6.56	8.13
pal	0.47	0.45	4.78	4.98
ls	0.003	0.004	0.05	0.05
gls	0.002	0.002	0.02	0.02
laah	0.002	0.002	0.18	0.20
pl	0.01	0.02	1.07	1.20

There is a notable change in results when we add an additional level of sophistication and check for the use of simple patterns: the averages in root projects are almost zero and remixing plays no part in improving the situation. The usage is 1% and less except for "Search" and "Process All Items" (pal).

Next, in Table X we show the distribution of projects based on their remix depth. To observe the effect of the depth of remixes, we measure the average LOC on the basis of remix depth. This is shown in Table XI.

TABLE X
DISTRIBUTION OF PROJECTS BASED ON REMIX DEPTH

Number of projects with undefined remix depth	2,975 (4.62%)
Number of projects with remix depth of one	48,420 (75.22%)
Number of projects with remix depth of two	6,383 (9.92%)
Number of projects with remix depth of three	4,714 (7.32%)
Number of projects with remix depth of above three but less than fifty	1805 (2.80%)
Number of projects with remix depth of above fifty	74 (0.11%)

It can be observed that majority of projects (75.22%) have one level of remix and the second major group is with 2 to 3 levels of remixes, with a combined percentage of 17.24%. So these two groups are of particular interest to us. The other groups will be used for exceptional cases analysis. Table XI also reveals that each addition of remix level leads to more LOC.

Next we show the use of patterns in remixed vs root projects and how they compare on the basis of level of remixes.

Figure 2 shows the overall picture of the average use of patterns and important programming constructs regardless of the remix level. Though small, it shows an improvement in remixed projects vs. root projects in some cases. However, the improvement appears to be minor in and nothing seems to have a big increase perhaps with the exception of "forever". It can also be noted that the use of six elementary patterns is very

TABLE XI
DISTRIBUTION OF PROJECTS BASED ON REMIX DEPTH

Description	Root	Remixed
LOC for projects with remix depth of one	284.18	294.96
LOC for projects with remix depth of two	583.57	601.82
LOC for projects with remix depth of three	201.05	230.43
LOC for projects with remix depth of above three but less than fifty	297.31	510.74
LOC for projects with remix depth of above fifty	96.82	991.84

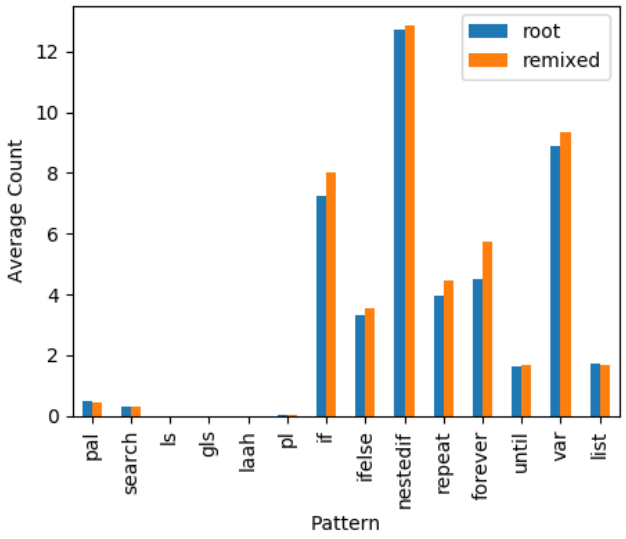

Fig. 2. Average number of patterns in all projects

low; three patterns are used occasionally (Process All Items, Search, Polling Loop), but the rest are nearly non existent.

For deeper analysis we divide the data on the basis of remix depth from root to the final version under consideration. This is shown in Figure 3, which shows similar trends, albeit with even smaller improvements. The use of "nestedif" is higher in root projects compared to remixed projects, and also lists appears to be decreasing slightly in usage in remixed projects. This indicates the effect on the average from a small number of projects with huge block count. A very important fact is that 75.22% of the projects have a remix depth of 1, so this is the most representative version of the overall picture.

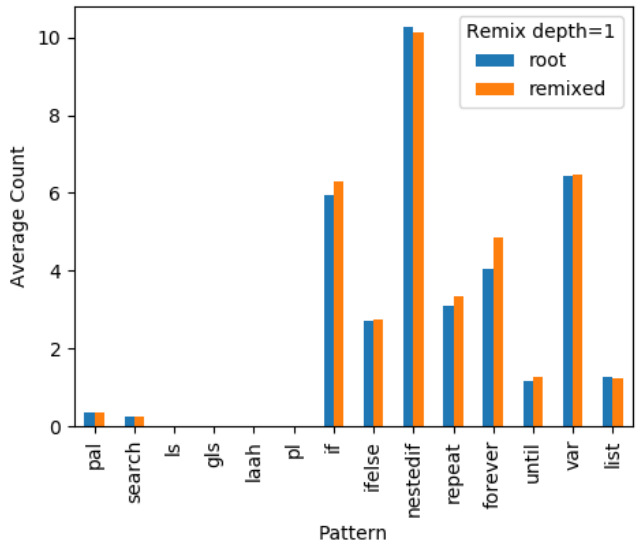

Fig. 3. Average number of patterns in projects with remix depth of 1

In Figure 4, where remixes that are 2 or 3 deep are shown, the improvements become more visible in some primitives while other patterns and programming elements are used less.

The figures 3 and 4 with a remix depth of 1 to 3 are the most representative, as it covers 95% of the projects. It is also clear that with each level of remix LOC tends to increase, which naturally increases the use of important programming constructs, but the use of patterns that show more sophisticated use of programming elements stays very low.

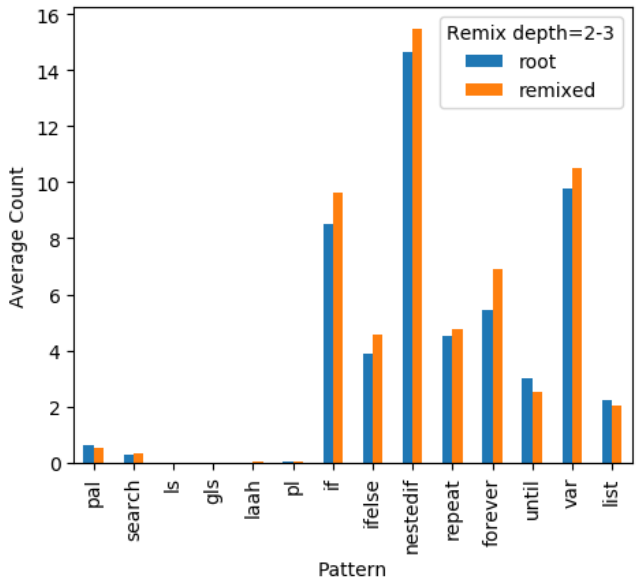

Fig. 4. Average number of patterns in projects with remix depth of 2 or 3

V. Intermediate Remixes

One important thing to consider is that for all projects having a remix depth of greater than one, there could be any number of remixes between the root and the project. Although the majority of projects (75.22%) have a remix depth of one and the analysis holds true for them, 17.24% of the projects have a remix depth of 2 or 3, and a further 2.91% have a remix depth greater than 3. This means there could be any number of remixes between the root and the project (and in one case, there were 476 remixes for a project!) Any of those remixes could completely change the appearance of the project compared with the root (e.g. remove all blocks and do nothing else) and the final version could be very different from the root version, which could impact the results above. To address this, here we analyze all the intermediate remixes between the root and the remixed projects to confirm whether this is a big issue or not.

There were 12,976 (20.15%) projects that had a remix depth of greater than one. We gathered the block counts (LOC) of the root and remixed projects and all of the intermediate remixes and calculated the standard deviation in block count for each series of remixes. For example, remixed project 135182346 has 3 intermediate remixes from the root (114033547). The

block count for the root project is 36 and for the final remixed project it is 115. The block counts for each of the intermediate remixes is 89, 106, and 96 respectively. The standard deviation for this list is 27.64. We did the same for all 12,976 projects and calculated the mean and median of the standard deviations, which came out to be 81.98 and 17.39 respectively. The median is perhaps more appropriate as it ignores the effect of a small number of large values. Given that the average number of blocks in root projects with remix depth over 1 was 400.23 and the average number of blocks in remixed projects with remix depth over 1 was 454.75, the standard deviation of 17.39 indicates that the change in block counts is not substantial compared with the average project size; that is, it appears that projects are usually only modified a little.

Next we compare the average gain (number of blocks in remixed projects vs number of blocks in root projects) in all projects that have a remix depth of 2 or more. The average gain for the set was +50.28, while the corresponding number for projects with remix depth of one was +10.78. It highlights that more remixes generally leads to a higher number of blocks, i.e. that students are usually adding code to projects.

These results show that the effect of remixes can be unpredictable and while it appears that in general projects have a small percentage of code added to them in a remix, the size often changes no more than just a few lines of code. But other than a small percentage of exceptional cases, the majority of projects show consistent trends that seem in line with the intent of having remixing available. While it is possible that the small change in block counts could reflect most of a project being deleted and new code added, a manual analysis of a sample of the projects also showed signs of remixing activity with the addition of new blocks consistent with the results.

VI. Conclusion

Remixing is popular in the Scratch community, and the aim of this study was to look at this activity objectively and provide a broad perspective. We measured block repertoire, block counts (LOC), and the use of elementary patterns in users' projects, comparing both their original work and remixed programs. We compared these measures to gain insight into the contribution of remixing to learning new CT concepts and programming techniques. We also investigated the progression through remixes of programming skills shown via LOC and the use of elementary patterns, based on tracking changes between root and remixed projects. To get a more detailed understanding, intermediate remixes were also counted for the projects that had a remix depth of greater than one. The results showed the adoption of some concepts from the program being remixed, while other concepts didn't seem to be picked up by remixing. From this the impact of remixing is not clear, and we cannot say for sure that remixing activity in Scratch promotes learning and helps progression. While there are some positive indicators of simple concepts being picked up, there seems to be little evidence of more sophisticated constructs being learned, and this supports the viewpoint that we need better pedagogy to teach programming to fully utilize the potential of block based programming in general, and remixing activity in particular.

Although this work provides a better understanding of the nature of projects carried out by Scratch users, and especially remixing based on (large) samples, to gain a better insight we would next want to track other elements of a user's learning journey, such as their use of teaching resources, in-person peer learning, and teacher direction.

References

[1] A. Monroy-Hernández, "Scratchr: Sharing user-generated programmable media," in *Proceedings of the 6th International Conference on Interaction Design and Children*, ser. IDC '07. New York, NY, USA: ACM, 2007, pp. 167–168. [Online]. Available: http://doi.acm.org/10.1145/1297277.1297315

[2] D. M. Wilkinson and B. A. Huberman, "Cooperation and quality in wikipedia," in *Proceedings of the 2007 International Symposium on Wikis*, ser. WikiSym '07. New York, NY, USA: ACM, 2007, pp. 157–164. [Online]. Available: http://doi.acm.org/10.1145/1296951.1296968

[3] A. Kittur and R. E. Kraut, "Harnessing the wisdom of crowds in wikipedia: Quality through coordination," in *Proceedings of the 2008 ACM Conference on Computer Supported Cooperative Work*, ser. CSCW '08. New York, NY, USA: ACM, 2008, pp. 37–46. [Online]. Available: http://doi.acm.org/10.1145/1460563.1460572

[4] B. Simon and Q. Cutts, "Peer instruction: a teaching method to foster deep understanding," *Communications of the ACM*, vol. 55, no. 2, pp. 27–29, February 2012.

[5] M. Diehl and W. Stroebe, "Productivity loss in brainstorming groups: Toward the solution of a riddle." *Journal of personality and social psychology*, vol. 53, no. 3, p. 497, 1987.

[6] K. Andrew, "The cult of the amateur: How todays internet is killing our culture," *Crown Business, Doubleday, Random House*, pp. 70–75, 2007.

[7] 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

[8] P. Techapalokul and E. Tilevich, "Understanding recurring quality problems and their impact on code sharing in block-based software," in *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Oct 2017, pp. 43–51.

[9] T. Bell, "Establishing a nationwide CS curriculum in New Zealand high schools," *Communications of the ACM*, vol. 57, no. 2, pp. 28–30, 2014.

[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] 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

[15] K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y. Ye, "Evolution patterns of open-source software systems and communities," in *Proceedings of the International Workshop on Principles of Software Evolution*, ser. IWPSE '02. New York, NY, USA: ACM, 2002, pp. 76–85. [Online]. Available: http://doi.acm.org/10.1145/512035.512055

[16] Y. Ye and K. Kishida, "Toward an understanding of the motivation open source software developers," in *Proceedings of the 25th International Conference on Software Engineering*, ser. ICSE '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 419–429. [Online]. Available: http://dl.acm.org/citation.cfm?id=776816.776867

[17] S. L. Huff, M. C. Munro, and B. Marcolin, "Modelling and measuring end user sophistication," in *Proceedings of the 1992 ACM SIGCPR Conference on Computer Personnel Research*, ser. SIGCPR '92. New York, NY, USA: ACM, 1992, pp. 1–10. [Online]. Available: http://doi.acm.org/10.1145/144001.144011

[18] M. C. Munro, S. L. Huff, B. L. Marcolin, and D. R. Compeau, "Understanding and measuring user competence," *Information Management*, vol. 33, no. 1, pp. 45 – 57, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0378720697000359

[19] C. Scaffidi, M. Shaw, and B. Myers, "An approach for categorizing end user programmers to guide software engineering research," in *Proceedings of the First Workshop on End-user Software Engineering*, ser. WEUSE I. New York, NY, USA: ACM, 2005, pp. 1–5. [Online]. Available: http://doi.acm.org/10.1145/1082983.1083232

[20] M. Shaw, A. Ko, C. Scaffidi, and B. Myers, "Dimensions characterizing programming feature usage by information workers," in *IEEE Symposium on Visual Languages and Human-Centric Computing(VLHCC)*, vol. 00, 09 2006, pp. 59–64. [Online]. Available: doi.ieeecomputersociety.org/10.1109/VLHCC.2006.21

[21] B. W. Boehm, Clark, Horowitz, Brown, Reifer, Chulani, R. Madachy, and B. Steece, *Software Cost Estimation with Cocomo II with Cdrom*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2000.

[22] 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

[23] 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.

[24] K. Brennan and M. Resnick, "New frameworks for studying and assessing the development of computational thinking," in *In AERA 2012*, 2012.