Software Clones in Scratch Projects
On the Presence of Copy-and-Paste in Computational Thinking Learning

Robles, Gregorio; Moreno-León, Jesús; Aivaloglou, Efthimia; Hermans, Felienne

DOI
10.1109/IWSC.2017.7880506

Publication date
2017

Document Version
Final published version

Published in
2017 IEEE 11th International Workshop on Software Clones (IWSC)

Software Clones in Scratch Projects: On the Presence of Copy-and-Paste in Computational Thinking Learning

Gregorio Robles[*], Jesús Moreno-León[†], Efthimia Aivaloglou[‡], and Felienne Hermans[‡]
[*]Universidad Rey Juan Carlos, Fuenlabrada (Madrid), Spain
grex@gsyc.urjc.es
[†]Programamos.es & Universidad Rey Juan Carlos, Spain
jesus.moreno@programamos.es
[‡]Delft University of Technology, The Netherlands
{e.aivaloglou, f.f.j.hermans}@tudelft.nl

***Abstract*—Computer programming is being introduced in schools worldwide as part of a movement that promotes Computational Thinking (CT) skills among young learners. In general, learners use visual, block-based programming languages to acquire these skills, with Scratch being one of the most popular ones. Similar to professional developers, learners also copy and paste their code, resulting in duplication. In this paper we present the findings of correlating the assessment of the CT skills of learners with the presence of software clones in over 230,000 projects obtained from the Scratch platform. Specifically, we investigate i) if software cloning is an extended practice in Scratch projects, ii) if the presence of code cloning is independent of the programming mastery of learners, iii) if code cloning can be found more frequently in Scratch projects that require specific skills (as parallelism or logical thinking), and iv) if learners who have the skills to avoid software cloning really do so. The results show that i) software cloning can be commonly found in Scratch projects, that ii) it becomes more frequent as learners work on projects that require advanced skills, that iii) no CT dimension is to be found more related to the absence of software clones than others, and iv) that learners -even if they potentially know how to avoid cloning- still copy and paste frequently. The insights from this paper could be used by educators and learners to determine when it is pedagogically more effective to address software cloning, by educational programming platform developers to adapt their systems, and by learning assessment tools to provide better evaluations.**

I. Introduction

In recent years programming is being promoted in schools around the world as part of a movement that aims to promote Computational Thinking (CT) skills. CT is the process by which a problem is formulated and a solution is expressed in such a way that a computer can carry it out effectively. It is based on an iterative process with three stages: the formulation of a problem (abstraction), the expression of a solution (implementation), and the execution and evaluation of the solution (analysis). Although the term CT has recently been popularized by Wing [1], it was already used by Papert in the 1980s [2].

There are many environments that help learners to acquire these skills, mostly based on visual languages that allow learners to implement their solutions by combining pre-defined blocks. Among these solutions we can cite Alice [3], Kodu [4] or Scratch [5], a framework with more than 15 million users and over 18 million projects shared in its open online repository[1].

In the same way as *professional* software developers use copy and paste as an act of *ad-hoc* reuse, we can find such behavior as well in learners. This behavior is harmful: a recent controlled experiment with first year high-school students found that the ones working on programs exhibiting code smells performed significantly worse. Especially for the presence of code clones, it was found that it decreases the students' ability to modify the Scratch programs [6].

Specifically, in this paper, we want to shed some light on software cloning in programming learning environments. In particular, we analyze a sample of over 230,000 Scratch projects to answer following research questions:

- **RQ1: How frequent is software cloning in Scratch projects?**
  The aim of this question is to see if software cloning is a common practice in Scratch projects. We compare this magnitude with the presence of custom blocks, which is the way functionality can be reused in Scratch programs. We expect, based on our experience teaching kids, to frequently find cloning in Scratch projects, more often than custom blocks.
- **RQ2: Is software cloning independent of the mastery required to create a project?**
  The goal of this question is to find out if software cloning occurs in Scratch projects of any level of complexity. If this is not the case, it would be interesting to see if it occurs in the early stages of the learning process (with projects that require less mastery) and then disappears while confronting more difficult projects, or even if it gets more present as projects become more complex.
  We expect that as the mastery score increases, cloning should disappear, as learners with higher development of skills will try to avoid clones.

[1]http://scratch.mit.edu

- **RQ3: Is software cloning related more to any of the CT dimensions?**
  This question addresses the fact that software cloning might have a stronger relation to some of the CT dimensions assessed by `Dr. Scratch`[2].
  We expect that some dimensions will be more related to cloning. For instance, we *naively* assume that the *data representation* dimension should be less affected by software cloning than the *parallelism* dimension, as while the former considers the use of variables and the types of variables, the latter requires the use of two scripts (and therefore the chances of having similar, cut-and-paste prone, structures is higher).
- **RQ4: Do learners who have the ability to avoid software cloning do it?**
  Scratch offers several ways for learners to avoid software cloning in their projects.
  We expect that once learners have developed the skills necessary to avoid cloning, software clones will not be found in those Scratch projects.

The structure of the paper is as follows: After providing some background information on related research, in Section III we present the research methodology used in the paper, in particular we explain how we have identified clones and assessed the mastery of Scratch projects. The next section introduces the case study, while Section V offers the findings of our study. Finally, we discuss our results, including limitations and possible paths for future research.

II. Background

A. Cloning in Scratch Manuals

The most used Scratch manual is *Creative computing* [13], a guide developed by members of the *ScratchEd* research team at the Harvard Graduate School of Education. This guide, which is available in eight languages, presents "a collection of ideas, strategies, and activities for an introductory creative computing experience using the Scratch programming language". Software cloning (and how to avoid it) is barely mentioned in this guide. When the feature to create new, user-defined custom blocks is introduced in unit 3, there is no discussion of code repetition. In addition, the explanation for this feature and the examples proposed do not include information on how to use parameters, which limits the potential of user created blocks to be utilized in different situations. In the last unit of the guide, dedicated to advanced concepts, the feature to create instances of sprites is introduced with the following, short explanation: "cloning is an easy way to create multiples of the same sprite. You can use cloning to make many objects and create cool effects in a project".

From reviewing other Scratch manuals, we have found no uniform criteria in terms of how and when to explain the concepts of software cloning and/or software reuse. Some of them even omit it. In *Scratch 2.0 Sams Teach Yourself in 24 Hours* [14], a book that includes 24 sessions of one hour that show how to create animations, simulations, stories and games, the feature to create instances of sprites is presented in unit 9, while the possibility of creating blocks is introduced in unit 15. In *Scratch Programming in Easy Steps* [15], an introduction to learn to create animations and games, no information appears on the use of instances of sprites, while custom blocks is shown in the very last chapter. Finally, in *Learn to Program with Scratch: A Visual Introduction to Programming with Games, Art, Science, and Math* [16], which is a collection of hands-on projects that are designed to teach concepts to solve real-world programming problems, one of the first chapters introduces the creation of custom blocks "to enforce good programming style from the beginning"; however, the creation of instances of sprites is not discussed in this book.

B. Cloning in Learning Environments

Cloning has been found to decrease the ability of novice programmers to modify Scratch programs in a controlled experiment with 61 first year high-school students in [6]. Also related to this work on clone identification is [12], which analyzes the same dataset of Scratch projects for various code smells, including code duplication. Code duplication is found in 26% of the projects, which is more than what we find in this work. This difference is attributed to the different definition for code clones: that analysis uses 5 as the minimum number of equal blocks in a sequence to be considered a clone, unlike this work where we have set this limit to 6 blocks. Moreover, while [12] also evaluates the projects in terms of utilized programming concepts, it does not analyze the relationship between code duplication and mastery or CT skills.

C. Assessment of CT Skills in Scratch

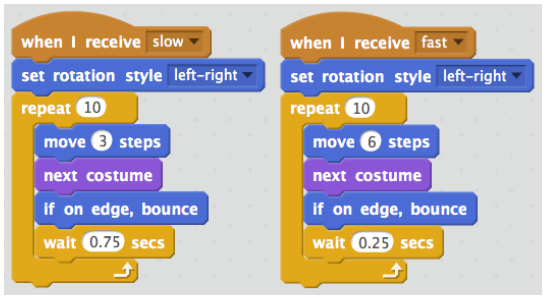
A number of studies have been carried out on assessing the programming skills of novice programmers through the analysis of Scratch programs for indications of learning of programming concepts. Maloney *et al.* [17] analyzed 536 Scratch projects for blocks that relate to programming concepts including loops, conditional statements, variables, user interaction, synchronization, and random numbers. Yang *et al.* examined the learning patterns of programmers in terms of block use over their first 50 projects [18]. In [19], the use of programming concepts was examined in relation to the level of participation, the gender, and the account age of five thousand Scratch programmers. Dasgupta *et al.* investigated how project remixing relates to the adoption of computational thinking concepts [20]. In [21], a model for assessing computational thinking in primary school students is proposed and applied on 150 Scratch projects, finding that design patterns requiring understanding of parallelism, conditionals and, especially, variables were under-represented until a certain age.

III. Methodology

A. Cloning in Scratch Projects

Hairball is a Scratch code analyzer that detects bad practices in Scratch programs, such as never executed code or non-initialized object attributes [7], [8]. Hairball is a plug-in based

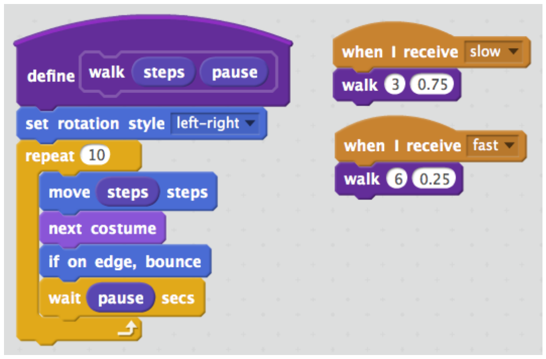
[2]http://www.drscratch.org



```
when I receive slow
set rotation style left-right
repeat 10
    move 3 steps
    next costume
    if on edge, bounce
    wait 0.75 secs

when I receive fast
set rotation style left-right
repeat 10
    move 6 steps
    next costume
    if on edge, bounce
    wait 0.25 secs
```

Fig. 1. Two scripts repeating code.



```
define walk steps pause
set rotation style left-right
repeat 10
    move steps steps
    next costume
    if on edge, bounce
    wait pause secs

when I receive slow
walk 3 0.75

when I receive fast
walk 6 0.25
```

Fig. 2. Definition of a custom block to avoid code repetition.

architecture, in which we added functionality to detect duplicated code[3], called `duplicate.DuplicateScripts` within a Scratch project. All blocks in a project are tokenized, so that two sequences that only differ in the receiving values are considered to be clones. The number of equal blocks in a sequence to be considered a *clone* and not coincidentally similar has been set to 5, i.e., the smallest possible sequence is composed of 6 blocks. The identification procedure is intra-project and not inter-project, so cloning between projects is not considered in this study.

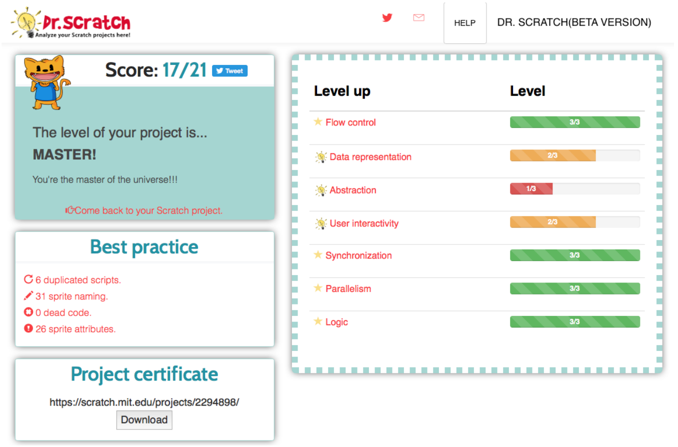
Figure 1 shows two programs containing blocks where the only difference lies in the values that they receive as parameters. When processed by the `duplicate.DuplicateScripts` plug-in, these programs are tokenized. In both cases, the code of the programs would be translated into the following sequence of tokens:

```
'when I receive %s'
'set rotation style %s'
'repeat %s%s'
'move %s steps'
'next costume'
'if on edge, bounce'
'wait %s secs'
```

This would result in the plug-in considering them a *clone*. The *right way* to implement this functionality in Scratch would be by defining a custom block that receives several values per parameter and reusing this block in both programs, as shown in Figure 2. The maintainability of this implementation would be higher. Custom blocks are sprite-specific, i.e., Scratch does not support the definition of project-wide custom blocks. Our plug-in takes this into consideration: it only considers software clones found in the same sprite.

Scratch offers a feature that users can use to avoid duplicating sprites. This feature, labeled *create clone* in the Scratch environment, allows to generate new instances of a character; each created instance executes the programs of the sprite. One of the plug-ins developed for the Hairball environment allows to detect the use of this feature in the source code of the projects. Aiming to avoid confusion, in this paper we refer to this feature as *instances of sprites*.

[3]The plug-in is publicly available at https://github.com/jemole/hairball/blob/master/hairball/plugins/duplicate.py



Dr.Scratch
Analyze your Scratch projects here!
HELP
DR. SCRATCH(BETA VERSION)
Tweet
Download

Score: 17/21

The level of your project is...

MASTER!

You're the master of the universe!!!

Come back to your Scratch project.

Best practice

- 6 duplicated scripts.
- 31 sprite naming.
- 0 dead code.
- 26 sprite attributes.

Project certificate

https://scratch.mit.edu/projects/2294898/

Level up	Level
Flow control	3/3
Data representation	2/3
Abstraction	1/3
User interactivity	2/3
Synchronization	3/3
Parallelism	3/3
Logic	3/3

Fig. 3. Dr. Scratch analysis for a project with advanced CT Score. The Scratch project is available at https://scratch.mit.edu/projects/2294898/

According to the classification in [10], in this paper we are identifying clones of Type-1 (“an exact copy without modifications”) and Type-2 clones (“a syntactically identical copy; only variable, type, or function identifiers were changed”), in the latter case –at least partially– as we do not take into consideration changes to function identifiers.

B. Assessment of CT Skills

`Dr. Scratch` is a free web-based tool that analyzes Scratch projects for the development of seven dimensions of the CT competence: abstraction and problem decomposition, logical thinking, synchronization, parallelism, algorithmic notions of flow control, user interactivity and data representation [9]. These dimensions are evaluated by means of a static analysis of the source code and are given a score that ranges from 0 to 3, following the criteria that are presented in Table I. The aggregated sum of all dimensions gives the mastery score, which is hence a number between 0 and 21. An example of an assessment report can be seen in Figure 3; the project analyzed has achieved 17 points of mastery score, pointing out an *advanced* CT level of its author.

Dr. Scratch has been conceived following the ideas of Scrape [11], a tool that allows visualizing the blocks being used in a Scratch project, and its core is built on top of Hairball [7].

TABLE I
LEVEL OF DEVELOPMENT FOR EACH CT DIMENSION ASSESSED BY DR. SCRATCH [9]

CT dimension	**Basic (1pt)**	**Developing (2pt)**	**Proficient (3pt)**
Abstraction and problem decomposition	more than one script and more than one sprite	procedures (creation of custom blocks)	use of clones (instances of sprites)
Parallelism	two scripts on green flag	two scripts on key pressed or on the same sprite clicked	two scripts on when I receive message, or video or input audio, or when backdrop changes to
Synchronization	wait	message broadcast, stop all, stop program	wait until, when backdrop changes to, broadcast and wait
Flow control	sequence of blocks	repeat, forever	repeat until
User interactivity	green flag	keyboard, mouse, ask and wait	webcam, input sound
Logical Thinking	if	if else	logic operations
Data representation	modifiers of object properties	variables	lists

IV. CASE STUDY

The dataset that we used for this study is the one made available from [12]. The dataset consists of 250,166 projects scraped from the Scratch project repository. Those are the projects most recently shared in the Scratch projects page[4] when the web scraping program run on March 2nd 2016. The web scraping program, as well as all scraped projects in JSON format are available.[5]

We have run the `duplicate.Duplicate` scripts on all the Scratch projects from our data set, storing in a CSV file the information on the number of clones and custom blocks found in each of the projects. In addition, we have run `Dr. Scratch` on all projects in order to obtain the scores for the seven dimensions and the total mastery score, and have added this information to the CSV file. The data in the CSV file is then analyzed with Pandas. The raw data and the IPython notebook of the analysis can be obtained in the replication package[6].

V. FINDINGS

A. RQ1: Frequency of Cloning in Scratch Projects

Out of the 231,050 projects that compose our sample, 46,653 (20.19%) contain at least one software clone. On the other hand, we can find 17,863 projects that define custom blocks (7.73%). This means that there are almost three times more projects with software clones than with custom blocks. We have identified as well 6,060 projects that contain both software clones and custom blocks; this supposes 2.62% of all projects. 12.99% of projects with clones have custom blocks, and 33.92% of the projects that create custom blocks contain software clones.

The correlation between cloning and custom blocks is 0.124, noting that –when considering the complete sample– we cannot infer that projects with clones are more prone to use custom blocks.

RQ1: Software clones can be commonly found in Scratch projects (*ca.* 20%), almost three times more frequently than custom blocks (*ca.* 7.5%). A third of the projects where user-created custom blocks are found also contain software clones.

[4]https://scratch.mit.edu/explore/projects/all/
[5]https://github.com/TUDelftScratchLab/ScratchDataset
[6]https://gsyc.urjc.es/~grex/repro/2016-iwsc-scratch

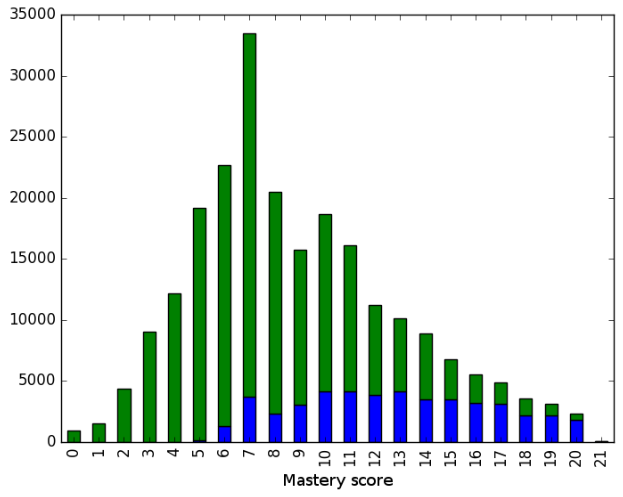


Fig. 4. Distribution of projects with clones (blue) and without clones (green) in terms of their mastery score.

B. RQ2: Software Cloning and Mastery Score

Figure 4 offers a histogram with the distribution of the total mastery score of the projects under analysis. As it can be seen the mastery score is not evenly distributed, as most of the projects are located in the range between 6 and 11 points. Only a minor number of projects have a very low or very high score. The figure also provides information on the projects with (blue) and without (green) software clones; the amount of projects that contain at least one clone is not relevant until a mastery score of 6, and from then on, the share of projects with clones increases with the mastery score. More than 52% the projects that have at least 16 points of mastery have at least one clone.

RQ2: Cloning and custom blocks are almost non-existent until a given mastery score (of 6/21). Cloning is then extensively present in Scratch projects and becomes more frequent as the mastery score rises. For projects of a relative complexity onwards (mastery score > 15), more than half of the projects contain software clones.

TABLE II
NUMBER OF PROJECTS WITH CUSTOM BLOCKS (CUSTOMB), AND WITH CUSTOM BLOCKS AND SOFTWARE CLONING (ABSOLUTE AND PERCENTAGE) BY MASTERY SCORE.

Mastery	w/CustomB	$w/CustomB \cap w/Clones$	$\%w/CustomB \cap w/Clones$
6	457	1	0.22%
7	1,046	17	1.63%
8	1,107	43	3.88%
9	929	74	7.97%
10	778	81	10.41%
11	837	117	13.98%
12	975	151	15.49%
13	1,147	208	18.13%
14	1,061	236	22.24%
15	1,112	306	27.52%
16	1,391	553	39.76%
17	1,589	796	50.09%
18	1,587	907	57.15%
19	1,671	1,053	63.02%
20	1,907	1,446	75.83%
21	89	71	67.62%

TABLE III
NUMBER OF PROJECTS WITH INSTANCES OF SPRITES (IOS), AND WITH INSTANCES OF SPRITES AND SOFTWARE CLONING (ABSOLUTE AND PERCENTAGE) BY MASTERY SCORE.

Mastery	w/IoS	$w/IoS \cap w/Clones$	$\%w/IoS \cap w/Clones$
6	12	0	0.00%
7	112	0	0.00%
8	371	30	8.09%
9	979	179	18.28%
10	1,246	284	22.79%
11	1,787	241	13.49%
12	1,648	407	24.70%
13	1,582	386	24.40%
14	1,910	518	27.12%
15	2,183	955	43.75%
16	1,555	729	46.88%
17	1,828	992	54.27%
18	2,470	1,328	53.77%
19	2,709	1,937	71.50%
20	2,339	1,841	78.71%
21	105	86	81.90%

C. RQ3: Software Cloning and CT Dimensions

Figure 5 shows the amount of projects with clones by the score obtained for each dimension, as provided by Dr. Scratch. As it can be seen, there is no dimension that is free of cloning. For lower scores of some dimensions the absence of software clones is related to the low complexity of the programs (as we have already seen for projects that require a total mastery below 6). However, for all of the dimensions around 50% of the projects with three points have software clones. This means that none of the current seven dimensions under evaluation serves as a way of mitigating the effect of copy-and-paste by learners.

RQ3: Software clones are present in all CT dimensions. In the highest level of developed skills of all dimensions, software cloning can be found in around half of the projects.

D. RQ4: Avoidance of Software Cloning by Advanced Learners

The results for the "Abstraction [and problem decomposition]" dimension (on the left in Figure 5) allow us to answer RQ4. In total numbers, there are 17,863 projects that incorporate custom blocks. Of these, 6,060 projects (33.92%) also include software clones.

Table II offers more insight into the common presence of custom blocks and software cloning. The trend for the overall mastery score (in Figure 4) is found to be replicated with custom blocks: software cloning tends to increase with the mastery required for the projects. However, while with mastery, over 40% of the projects can be found to have software cloning from a score of 13 onwards, in projects containing custom blocks this happens for projects with at least a mastery score of 17. In other words, although the use of custom blocks does not mitigate software cloning, it delays its appearance towards more complex projects.

Table III provides the results from analyzing the common presence of instances of sprites and software cloning. Again, the trend is similar as the previous ones for the total mastery and custom blocks, with an increasing share of software cloning with the mastery score. As can be seen, the use of this feature delays the appearance of software cloning, although it does not completely avoid the issue, since over 40% of projects using instances of sprites with 15 mastery points or more present software cloning. Comparing these results with those in Table II, we can see that the use of instances of sprites has a lesser impact on the reduction of software cloning than custom blocks.

TABLE IV
CORRELATION COEFFICIENTS OF MASTERY SCORE, AS PROVIDED BY DR. SCRATCH, WITH THE PRESENCE OF CLONES, OF CUSTOM BLOCKS (CUSTOMB), AND OF INSTANCES OF SPRITES (IOS) IN SCRATCH PROJECTS.

	Clones	CustomB	IoS
Mastery	0.465	0.099	0.191

Table IV provides the correlation coefficients between the mastery score given by `Dr. Scratch` and the presence of clones (0.465), custom blocks (0.099) and instances of sprites (0.191). In all cases the relationship is positive, but the coefficient for clones is over twice as high as the one for instances of sprites, and the latter is twice as high as the one for custom blocks; this means that learners tend, while they become more proficient in Scratch, to use far more clones than to create instances of sprites and custom blocks.

RQ4: Learners who have the abilities to avoid software clones, also clone. However, they clone less frequently with the exception of very complex projects, where they clone the same.

VI. DISCUSSION

We have found that software cloning is an extended practice in the Scratch learning environment. Learners start to clone when in the *development* level, but make heavy use of cloning

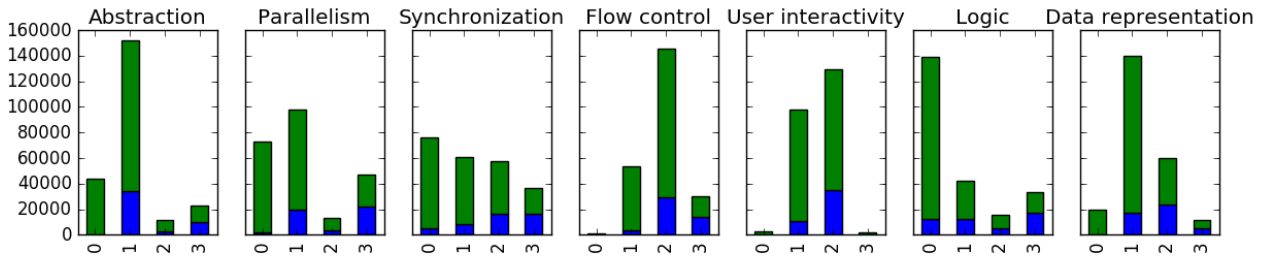


Fig. 5. Distribution of projects with clones (blue) and without clones (green) for each CT dimension assessed by Dr. Scratch.

during the *master* level. Although we expected cloning to disappear with high mastery scores, the results show that this does not happen.

In contrast, custom blocks are used far less than cloning (around one third). We expected that the use of custom blocks would make software cloning less used, but results show the opposite. At a mastery score of 10 is where the gap between cloning and custom blocks is highest. Probably it is at that point where it makes sense to pedagogically address cloning with learners, as at that point they have the necessity to have similar functionality several times in their project, but they do not know how to implement it. However, as Scratch currently does not support project-wide custom blocks, learners sometimes have no other option than to copy and paste custom blocks in all sprites that need them. Note that as we only consider intra-sprite clones in our study, this does not affect our results.

We argue that learning platforms, such as Scratch, should have a way to make learners give custom blocks preference to cloning. It is our opinion that by creating custom blocks, learners are acquiring higher skills in the resolution of problems, offering a more generalized solution that is easier to understand and maintain. This argumentation is inline with the goals of CT development in learners. We therefore argue that learning environments such as Scratch should include functionality to detect code cloning, and warn learners, advising to use custom blocks, for instance.

We also have found that cloning has not been paid the importance it should have, although some advances can be counted in the last years. So, in Scratch 1.4 there was no possibility of creating custom blocks or instances of sprites; these were introduced with Scratch 2.0 in May 2013. However, the most-used Scratch manuals do not sufficiently address the problem (and how to solve it with user-made custom blocks). On the other hand, the CT educational community has not taken it into consideration for their assessment. As a matter of fact, for the creation of the `Dr. Scratch` rubric, we analyzed how educators performed their assessments manually; software cloning was not part of any assessment, so it was not included in `Dr. Scratch`'s assessment. As by now, custom blocks contributes to 2 points for abstraction, while the creation of sprite instances adds 3 points, as experts thought that the latter requires a higher level of abstraction skills; if we consider our results (especially, the ones from RQ4), the assignment of points could be switched, as custom blocks address software cloning more than the creation of sprite instances (see the share of software clones for the *Abstraction* dimension in Figure 5). In our opinion, given that to achieve a *mastery* level of CT a proper use of custom blocks and cloning should be made, we should value a change in the assessment rubric of `Dr. Scratch` to specifically address the appearance of software cloning by introducing a penalization score when found.

A. Threats to Validity

The number of projects under study is low, as it represents around 1.5% of the total amount of Scratch projects that are hosted in the Scratch repository. Even though the sample is large, with over 230,000 projects, it could well be that it is not representative of the whole population of projects.

In our quest to find clones, we only identify some of the types of possible software clones (Type-1 and partially Type-2 clones). The other types of clones that can be found in the literature, Type-3 ("a copy with further modifications; statements were changed, added, or removed" [10]) and Type-4 ("two or more code fragments that perform the same computation but are implemented by different syntactic variants" [22]) are not considered. This means that we could potentially find more clones than the ones found, i.e., the amount of clones we report is a lower bound of the total number of clones. In this regard, the problem of software cloning in Scratch projects could be even worse than the one reported in this paper.

We only report intra-project cloning. The Scratch platform offers the possibility to *remix* projects from other users, an action that is similar to what is known as *forking* in well-known development platforms such as GitHub. Given that there are no libraries in Scratch, there are few possibilities to avoid inter-project cloning, making this a minor threat to the validity of our study. However, what can impact the results is the possibility that learners might have *remixed* a complex project without software clones, and then added these, resulting in a project with high mastery score and software clones.

In this paper, we have assumed software cloning as a bad practice, although there is evidence that in some contexts this has not to be the case [23]. However, of all the patterns of cloning described in [23], in our opinion only "Parameterized Code" ("When implementing a solution to a common problem, it is often the case that this solution can be modified to solve a new problem by changing only a few identifiers or literals in the code. This commonly occurs when implementing basic solutions for very similar problems, such as opening a file descriptor") and "Replicate and Specialize" ("[developers] may find code in the software system that solves a similar problem to the one they are solving. However, code may not be the exact solution, and modifications may be required. While the developer could generalize the original code, this may have a high cost in testing and refactoring") would apply to the Scratch environment; our intuition is that it would do in a very limited way.

The findings in our paper are Scratch-specific and may not be generalized to other block-based learning programming languages. There may be elements in how Scratch is conceived that *pushes* Scratch learners to cloning instead of to create custom blocks.

VII. Conclusions and Further Research

We have found software cloning to be frequently used in the Scratch learning environment. We show that it can be found independently of the skills of the learner and that it is independent of the CT dimensions that a learner develops. Even those learners who know how to potentially avoid software cloning, do it, although to a lesser extent until projects show certain complexity, where these learners show a similar pattern than the rest.

The insights from this paper could be used by educators and learners to determine when it is pedagogically more effective to address software cloning, by educational programming platform developers to adapt their systems, and by learning assessment tools to provide better evaluations.

Further research should be devoted to replicate this study with other, block-based programming learning environments to verify if cloning is widely used there as well. In addition, other types of clones in learning environments such as Scratch could be investigated. Other *bad smells*, such as long methods [6], could be also researched in a similar fashion to software cloning, as it has been done in this paper, to see if they as well are not addressed conveniently and do not disappear as the mastery of learners increases.

All in all, as researchers and educators it is our task to find out how to address software cloning in learning environments. As by now, and from our results from analyzing over 230,000 Scratch projects, the results show that there is a lot of room for improvement.

Acknowledgments

The first two authors are funded in part by the Region of Madrid under project "eMadrid: Investigación y Desarrollo de tecnologías educativas en la Comunidad de Madrid" (S2013/ICE-2715). The first author also acknowledges the SENECA project (MSCA-ITN-2014-EID).

References

[1] J. M. Wing, "Computational Thinking," *Communications of the ACM*, vol. 49, no. 3, pp. 33–35, 2006.

[2] S. Papert, *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc., 1980.

[3] S. Cooper, W. Dann, and R. Pausch, "Alice: a 3-d tool for introductory programming concepts," *Journal of Computing Sciences in Colleges*, vol. 15, no. 5, pp. 107–116, 2000.

[4] M. MacLaurin, "Kodu: End-user programming and design for games," in *Proceedings of the 4th international conference on foundations of digital games*. ACM, 2009, p. 2.

[5] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai, "Scratch: Programming for all," *Commun. ACM*, vol. 52, no. 11, pp. 60–67, Nov. 2009.

[6] F. Hermans and E. Aivaloglou, "Do code smells hamper novice programming? a controlled experiment on Scratch programs," in *Program Comprehension (ICPC), 24th International Conference on*. IEEE, 2016, pp. 1–10.

[7] 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*, ser. SIGCSE '13. New York, NY, USA: ACM, 2013, pp. 215–220.

[8] D. Franklin, P. Conrad, B. Boe, K. Nilsen, C. Hill, M. Len, G. Dreschler, G. Aldana, P. Almeida-Tanaka, B. Kiefer, C. Laird, F. Lopez, C. Pham, J. Suarez, and R. Waite, "Assessment of computer science learning in a Scratch-based outreach program," in *44th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '13. New York, NY, USA: ACM, 2013, pp. 371–376.

[9] J. Moreno-León, G. Robles, and M. Román-González, "Dr. Scratch: Automatic analysis of Scratch projects to assess and foster Computational Tshinking," *RED. Revista de Educación a Distancia*, vol. 15, no. 46, 2015.

[10] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo, "Comparison and evaluation of clone detection tools," *IEEE Transactions on Software Engineering*, vol. 33, no. 9, pp. 577–591, 2007.

[11] U. Wolz, C. Hallberg, and B. Taylor, "Scrape: A tool for visualizing the code of Scratch programs," in *Poster presented at the 42nd ACM Technical Symposium on Computer Science Education*, Dallas, TX, 2011.

[12] E. Aivaloglou and F. Hermans, "How kids code and how we know: An exploratory study on the Scratch repository," in *2016 ACM Conference on Intl Computing Education Research*. ACM, 2016, pp. 53–61.

[13] K. Brennan, C. Balch, and M. Chung, *Creative Computing*. Harvard Graduate School of Education, 2014.

[14] T. Warner, *Scratch 2.0 Sams Teach Yourself in 24 Hours*. Sams Publishing, 2014.

[15] S. McManus, *Scratch Programming in Easy Steps*. In Easy Steps Limited, 2013.

[16] M. Marji, *Learn to Program with Scratch: A Visual Introduction to Programming with Games, Art, Science, and Math*. No Starch Press, Inc., 2014.

[17] J. H. Maloney, K. Peppler, Y. Kafai, M. Resnick, and N. Rusk, "Programming by Choice: Urban Youth Learning Programming with Scratch," in *39th SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '08. ACM, 2008, pp. 367–371.

[18] 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 *Second ACM Conference on Learning @ Scale*. ACM, 2015, pp. 131–140.

[19] 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*, ser. WiPSCE '14. ACM, 2014, pp. 2–11.

[20] S. Dasgupta, W. Hale, A. Monroy-Hernández, and B. M. Hill, "Remixing as a pathway to Computational Thinking," in *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

[21] L. Seiter and B. Foreman, "Modeling the learning progressions of computational thinking of primary grade students," in *Ninth Annual International ACM Conference on International Computing Education Research*, ser. ICER '13. New York, NY, USA: ACM, 2013, pp. 59–66.

[22] C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," *Science of Computer Programming*, vol. 74, no. 7, pp. 470–495, 2009.

[23] C. J. Kapser and M. W. Godfrey, ""cloning considered harmful" considered harmful: Patterns of cloning in software," *Empirical Software Engineering*, vol. 13, no. 6, pp. 645–692, 2008.