Journal of Universal Computer Science, vol. 22, no. 12 (2016), 1533-1557
submitted: 27/3/16, accepted: 21/11/16, appeared: 1/12/16

Examining the Relationship between Socialization and Improved Software Development Skills in the Scratch Code Learning Environment

Jesús Moreno-León
(Programamos.es, Seville, Spain
jesus.moreno@programamos.es)

Gregorio Robles
(Universidad Rey Juan Carlos, Madrid, Spain
grex@gsyc.urjc.es)

Marcos Román-González
(Universidad Nacional de Educación a Distancia, Madrid, Spain
mroman@edu.uned.es)

Abstract: In the last years, socialization of the software development process has been proven to be an emergent practice, becoming social development platforms (such as GitHub or GitLab) very popular among software developers. However, little is still known about how social factors influence software development. In particular, in this paper we focus on how socialization affects the learning of programming skills, as developing software can be considered, in part, a continuous learning process. Aiming to shed some light in this regard, we analyze the social interactions of almost 70,000 users and the sophistication of over 1.5 million software products authored by them in the `Scratch` platform, the most popular social coding site for learning to program. The results indicate that there is a relationship between the social conducts of users and the improvement of their programming abilities, showing that more social actions performed by users is positively associated with more sophistication in their programs. Furthermore, the results also provide evidence that the relationship of social factors with the development of software programming skills tends to grow with time.

Key Words: social software development, social learning, computational thinking, Scratch
Category: K.3.2, D.2.3, D.2.6

1 Introduction

In the last years we are witnessing a movement that places socialization as an important factor in software development. Thus, there are studies that analyze the impact of social aspects on the way software ecosystems evolve over time [Mens and Goeminne, 2011], on how social processes can be accounted for the variations in software product quality [Sawyer and Guinan, 1998], on the use of microblogs in software development [Bougie et al., 2011], on how developers collaborate in knowledge sharing sites [Surian et al., 2010, Vasilescu et al.,

2013] and, especially, on the interactions among developers in social coding sites, such as GitHub [Thung et al., 2013, Jiang et al., 2013, Vasilescu et al., 2013]. These social coding sites, which integrate social media features with source code management tools, are gaining more and more popularity among software developers, as they provide a collaborative environment with high levels of social transparency [Pham et al., 2013].

Furthermore, given the volatile environment in which software is developed, with constant technological and organizational changes, learning-focused practices are to be considered as an essential part of software development [Hoda et al., 2013]. Nonetheless, some development methodologies, such as the *agile* ones, consider that software development is a continuous learning process [Babb et al., 2014, Augustine et al., 2005] that involves different types of learning: "learning new or complex technical skills, learning cross-functional skills, and learning from the team's own experiences, all of which fuel self-improvement." [Hoda et al., 2011]

However, there is a lack of studies that analyze how social factors affect the learning of software development skills. With this paper we want to shed some light on this regard by analyzing the social interactions of over one million users and the evolution of the software products that learners generated during five years in the most popular social coding site for programming learning, the `Scratch` [Resnick et al., 2009] platform. This site incorporates features to share, study and remix projects, post comments or work in teams, offering `Scratch` programmers the opportunity to learn the social aspects of software development.

It is noteworthy that GitHub, the most popular social coding site, and the `Scratch` platform are targeted to different audiences. The former is focused on professional software developers, while the latter has been designed for learners. However, most of the social functionalities can be found in both platforms: they offer the possibility to explore the repository of projects, to follow the activity of other users and projects, to express appreciation for a project, and to create *forks* (*remixes* in the `Scratch` jargon) of projects developed by other programmers. The `Scratch` platform and GitHub differ in that the former lacks support for software evolution, as the bug tracking, code review and pull-request features offered by GitHub can only be (partially) achieved through comments.

The remainder of this paper is structured as follows: the research goals are described in [Section 2]. Related research, focused on social aspects of software development, is reviewed in [Section 3]. [Section 4] presents the dataset utilized for the analysis, which includes the activities of over a million users during the first five years of `Scratch`. Results are presented in [Section 5]. [Section 6] gives an answer to the research questions addressed in the paper, discusses implications and points out the threats to the validity of our results. Finally, conclusions along with ideas for future research are outlined in [Section 7].

2 Research Goals

This work aims to identify the impact of social participation on the development of software development skills, by performing a comprehensive study of a social coding community for learning, `Scratch`. Specifically, the research questions that we address are following:

RQ1: Is there a relationship between learners' social participation in the `Scratch` community and the improvement in software development skills?

Actively participating in a social coding community is considered to help programming learners to know the social aspects of software development, which, in turn, is assumed to help them develop their coding skills. Aiming to measure the effect size of social conducts on software development skills, we will analyze potential differences in the improvement of project sophistication between users who actively participate in the community and those who do not make use of these social features. Moreover, we will establish levels of socialization to examine possible variations in the effect size of social conducts on software development skills.

RQ2: Is there a relationship between learners' remix conducts in the `Scratch` community and the improvement in software development skills?

Reusing code is assumed to be a good learning practice for programming learners, as it implies the reading, understanding and modification of other, commonly more experienced, developers' source code [Haefliger et al., 2008, Gross et al., 2010]. As `Scratch` offers the possibility of reusing code by means of remixing projects, we will study whether there are differences in the improvement in programming skills between those users who remix other programmers' projects and those who do not perform forks. Furthermore, the effect size of the remix conducts on software development skills could be compared with the effect size of the social aspects analyzed in RQ1.

RQ3: How does time spent in the community affect the answer to questions RQ1 and RQ2?

Our hypothesis is that, depending on the learning time, the relationship of social and remix actions with the development of programming skills may vary. Thus, we think that it is possible that participating very actively in the community might not be as beneficial for novice learners as for more experienced ones, or it could even be counterproductive for users who have just started to learn to program. In consequence, we will study whether there are differences in the effect size of both social and remix conducts on software development skills over time.

3 Related Research

Social software engineering is a relatively new, but important research area in software engineering. The idea of social software engineering –described as a complex socio-technical activity [Clarke et al., 2015], due to the need for sharing knowledge and discussing ideas among team members– was addressed by Ahmadi et al. by providing a literature survey in related areas [Ahmadi et al., 2008]. Storey et al. advocate for research to understand the benefits and risks of using social media tools and the effects they may have on the quality of software [Storey et al., 2010]. Based on a set of interviews with *light* and *heavy* users of GitHub, a series of social inferences that users make from the networked activity information of the site has been presented by Dabbish et al. [Dabbish et al., 2012]; respondents stated that they infer other developer's technical goals and vision when they edit code, or make guesses on the possibilities of success of similar projects, and that they combine these "inferences into effective strategies for coordinating work, advancing technical skills and managing their reputation".

The study of social activities in software development comprises research on micro-blogging [Bougie et al., 2011], on the use of Q&A sites such as StackOverflow [Surian et al., 2010, Vasilescu et al., 2013], or on social coding platforms such as GitHub [Thung et al., 2013, Jiang et al., 2013, Vasilescu et al., 2013], among others. The research topics are also very wide, covering collaboration [Surian et al., 2010], networking [Thung et al., 2013], information flow [Bougie et al., 2011, Vasilescu et al., 2013, Jiang et al., 2013], software evolution [Mens and Goeminne, 2011], etc. Some investigations on how to determine the development skills of developers can also be found. So, Capiluppi et al. have identified challenges of doing this for GitHub developers, and provide some guidelines to be used by recruiters to evaluate potential candidates [Capiluppi et al., 2013].

If we focus on the `Scratch` community, social aspects are clearly identified as key elements of the learning of programming abilities. Hence, Mitchel Resnick, the `Scratch` project leader at MIT Media Lab, states that "[m]embers of the `Scratch` community learn to collaborate in many different ways. They give feedback through comments on projects, they work together on joint projects, they remix one another's projects, they crowd-source artwork for their projects, they create `Scratch` tutorials to share their knowledge with one another" [Resnick, 2012], presenting three examples of collaborative learning and development in the `Scratch` community. Similarly, the `Scratch` *team* states that "[p]articipation and collaboration within online communities can support, inspire, and enable young people to become active creators (and not just consumers) of interactive media." [Brennan et al., 2010], although this statement is based on three concrete cases that illustrate different types of collaboration that allowed more complex and elaborated projects. In a similar vein, the `Scratch` *team* argues that *scratchers* "not only learn important math and computer science concepts,

but they also develop important learning skills: creative thinking, effective communication, critical analysis, systematic experimentation, iterative design, and continual learning" [Monroy-Hernández and Resnick, 2008], based on five cases in which user collaboration fostered the improvement of projects.

Most of the research papers that highlight the advantages for learners of participating in a social coding learning site are based on small case studies. However, some investigations on participation patterns in the `Scratch` community use larger amounts of data. In this regard, the contributions, mechanisms of gratification and patterns of participation of 65 young students (between 9 and 17 years old) learning to program with `Scratch` have been studied by Zuckerman et al. [Zuckerman et al., 2009]. These patterns revealed two types of participants: project creators and social participants. In addition they identified a long tail distribution in content contribution, with active and very active users (18% of the sample) creating 67% of the projects generated during the investigation.

100 randomly selected projects, along with their associated comments, were analyzed by Dahotre et al. in order to evaluate how `Scratch` is serving as a basis for demonstrating technical, social, and remixing skills [Dahotre et al., 2010]. Although authors acknowledge the success of the platform towards these goals, they also identify some opportunities for improvement. Thus, in spite of the important number of generated comments, researchers could not find a single case in which the feedback originated some kind of online collaboration. On the other hand, the level of code reuse was around 5%, significantly lower than the 15% stated by the `Scratch` *team*. In addition, only 40% of the remixes implied modifications in the code, being the majority of changes to modifying or including images and sounds.

Aiming to assess the evolution of `Scratch` users, both from technical and social points of view, data from 250 random users who had authored around 1,000 projects [Scaffidi and Chambers, 2012] was retrieved. Even though a positive progression in terms of social skills is detected, a negative progression in the technical abilities is also observed. It must be noted, though, that these two variables were independently analyzed, and correlation between them was not studied. Researchers state that the downtrend in the technical skills might be partially explained by the fact that experienced users may be less likely to demonstrate their skills than inexperienced users.

The social activities (downloads, comments, friends and favorites) of 2,225 users who at least created 1 project across a three-month period in early 2012 are analyzed by Fields et al. [Fields et al., 2014], identifying five classes of `Scratch` users in terms of their participation in the community: low network, downloaders, commenters, networkers and high network. In general, no correlation between level of online participation and level of programming sophistication is detected. However, the authors highlight that a very small and extremely active group of

users used more complex programming instructions than the rest.

With the goal of presenting implications in the design of collaborative tools and communities [Fields et al., 2013], the activity in the community to identify patterns of participation is analyzed. Velasquez et al. studied the comments generated by 5,000 users in a month, stating that comments about projects show a richer language than other kind of comments [Velasquez et al., 2014].

Finally, three recent investigations have been published based on the same dataset used in this paper. On one hand, the authors of this work have performed a preliminary investigation to measure how social the `Scratch` community is by studying user contributions in terms of number of comments, friends, favorites and galleries. The analysis involved over one million learners who had authored almost two million projects. The results indicate that the vast majority of users barely make use of the social capabilities offered by the `Scratch` platform [Moreno-León et al., 2016b]. On other hand, Matias et al. [Matias et al., 2016] have revisited the research by Scaffidi and Chambers [Scaffidi and Chambers, 2012], obtaining opposite results as `Scratch` users tend to share increasingly code-heavy projects over time and use a more diverse range of code concepts. In a similar work to the latter, the study of the whole repository shows that users who remix more often have larger repertoires of programming commands [Dasgupta et al., 2016].

Paper	Topic	Sample size
[Resnick, 2012]	Collaborative learning	3
[Brennan et al., 2010]	Participation in the community	3
[Monroy-Hernández and Resnick, 2008]	Collaborative learning	5
[Zuckerman et al., 2009]	Participation in the community	65
[Dahotre et al., 2010]	Collaborative learning, remixing, technical abilities	100
[Scaffidi and Chambers, 2012]	Participation in the community, technical abilities	1,000
[Fields et al., 2014]	Participation in the community	2,225
[Fields et al., 2013]	Participation in the community	5,000
[Matias et al., 2016]	Technical abilities	643,246
[Moreno-León et al., 2016b]	Participation in the community	2,000,000
[Dasgupta et al., 2016]	Remixing, technical abilities	2,400,000

Table 1: Summary of reviewed literature on `Scratch`: main topic and size of sample (in number of projects).

Table 1 summarizes the type of investigations that have been performed on projects shared in the Scratch community. As can be seen, a majority of investigations involve a small amount of projects with a sample of 5,000 projects or less for 8 out of the 11 papers. Articles placing the focus on the technical abilities of learners are a minority, as most of the works focus on other aspects of learning, being the most frequent subject of study the participation in the community.

4 Data Set

For this investigation we have worked with several datasets that include the first five years, approximately from 2007 to 2012, of public data from the Scratch online community website[1].

The set of datasets is divided into *Core datasets*, which describe the major objects and relationships captured by the Scratch website, including the main information on users, projects, galleries or favoriters; *Text and Code datasets*, which contain text generated by the programmers; and *Project Analytics Datasets*, such as the *project blocks* table, which holds the blocks used in each project.

In our study we have combined and analyzed data included in the *Core datasets* and *Project Analytics Datasets*. Specifically, we have worked with the information on 1,056,951 users, 1,928,699 projects, 120,097 galleries, 1,313,200 friends, 1,041,387 favorites and 7,788,414 comments on projects.

Although there is more than 1 million active users in the dataset, only 304,793 have published at least one project. As we are interested in comparing differences in learning and progression, we have selected those learners who have published at least 5 projects to study their social behavior. This process resulted in a new subset with the information of 67,799 users as presented in Table 2.

In a previous investigation, Scaffidi and Chambers [Scaffidi and Chambers, 2012] adapted a model proposed by Huff et al. [Huff et al., 1992] to measure the *sophistication* of Scratch projects. The authors derive sophistication from three different concepts: *breadth*, *depth* and *finesse*. *Breadth* refers to the range of different features that programmers use; *depth* represents the amount with which programmers use those features; while *finesse* captures the user's ability to solve programming problems effectively and creatively.

From the information in the datasets we could directly assign *blocks* to *depth* and *block.types* to *breadth*. To calculate *finesse*, instead of using the proposed framework by Scaffidi and Chambers [Scaffidi and Chambers, 2012], we have adapted Dr. Scratch, a free/libre static code analyzer for Scratch projects, to work with the data stored in the *project blocks* table.

[1] Access to the datasets can be requested at https://llk.media.mit.edu/scratch-data/

Field	Description
user.id	User id in the `Scratch` community
first	Date of the first published project
last	Date of the last published project
time	Days between the first and the last published project
number.projects	Amount of published projects
blocks.mean	Mean of the number of blocks (`Scratch` programming instructions) in the projects
block.types.mean	Mean of the number of types of blocks in the projects
favorited	Amount of projects marked as favorite by the user
galleries.created	Amount of galleries created by the user
friends	Amount of friends made by the user
pcomments	Amount of comments posted in projects by the user
forks	Amount of remixes (forks) created by the user
forks.code	Amount of remixes (forks) created by the user in which there are modifications in the code of the project
blocks.first	Number of blocks in the first published project
blocks.max	Maximum of number of blocks in all published projects
blocks.improve	Difference between *blocks.first* and *blocks.max*
block.types.first	Number of type of blocks in the first published project
block.types.max	Maximum of number of types of blocks in all published projects
block.types.improve	Difference between *block.types.first* and *block.types.max*

Table 2: Description of the information available for each user involved in the investigation

The effectiveness of assessing computational thinking skills using `Dr. Scratch`, which is based on `Hairball` [Boe et al., 2013] and inspired by `Scrape` [Wolz et al., 2011], has been previously reported [Moreno-León et al., 2015]. In addition, given that the assessment provided by `Dr. Scratch` is, to some degree, a complexity value, its measurements have been compared with other, classic software engineering metrics that measure the complexity of a program, such as McCabe's Cyclomatic Complexity [McCabe, 1976] and Halstead's metrics [Halstead, 1977], finding a statistically significant, positive, moderate correlation [Moreno-León et al., 2016a].

`Dr. Scratch` requires access to the source code file to be able to perform a complete analysis, which is not the case in this research. Consequently, we have performed some modifications in the analysis algorithm of `Dr. Scratch` to assign a mastery score, which ranges from 0 to 10 points, by studying the blocks included in the projects in terms of user interactivity, synchronization, logic,

data representation, flow control and other advanced instructions, as described in Table 3[2].

Category	Blocks adding 1pt	Blocks adding 2pts
Logic	if, if else	logic operations (and, or, not)
Data representation	variables	lists
Synchronization	wait, touching	message passing
Flow control	forever, repeat	forever if, repeat until, wait until
User interactivity	keyboard, mouse, ask & answer	
Other features	timer, random numbers	

Table 3: Description of the mastery score measurement, which is calculated based on the types of blocks used in the project. The score ranges from 0 to 10 points.

By performing the mastery analysis, we could add the information described in Table 4 to the dataset for the 1,539,197 projects finally under study.

Field	Description
mastery.first	Level of mastery demonstrated in the first published project
mastery.max	Maximum of level of mastery demonstrated in all published projects
mastery.improve	Difference between *mastery.first* and *mastery.max*

Table 4: Description of the information on *finesse* generated for each user by performing the mastery analysis to the projects in the dataset.

5 Results

Our dataset allows to study relationships between social and remix conducts with improvements in programming skills. The social contributions are analyzed in terms of number of projects marked as favorite, amount of galleries created, number of friends made and amount of comments posted in projects. Regarding

[2] The program used to perform the mastery analysis is publicly available at `http://gsyc.urjc.es/~grex/repro/2016-jucs-socialization`

the remix conducts, although there is information available on the total number of forks, we considered to take only into account those forks where the source code of the original project is modified. Finally, improvements are calculated in number of blocks, number of types of blocks and mastery score.

Figure 1 graphically represents the distribution of users in terms of time in the community, which is calculated as the number of days between the publication of the first and the last project of each user. As can be seen (the logarithmic scale of the vertical axis should be taken into account), the distribution is extremely skewed, with a vast majority of users with a short time in the community, and an extremely small set of users who spend a long time in the `Scratch` community exists. In consequence, time was considered a moderator variable that could affect the relationships between the independent variables, social and remix conducts, and the dependent one, the improvements in programming skills.

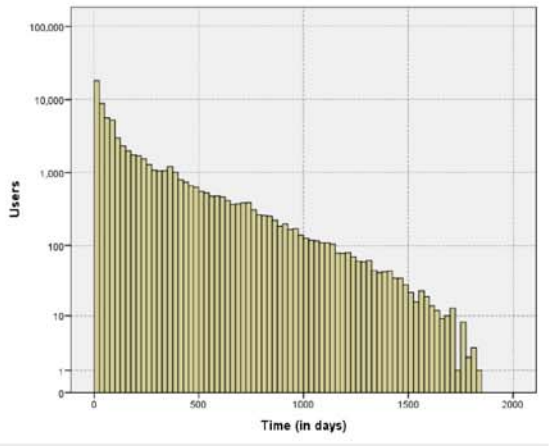

Figure 1: Distribution of users in terms of time (days) in the community. The vertical axis is in logarithmic scale.

5.1 Relationships between social conducts and improvements in programming skills

As a first approach to study whether social conducts are related to improvements in programming skills, we divided the users into those who perform any of the social actions and those who don't (see Table 5).

Table 6 shows the differences in improvements for each of the dichotomized groups, presenting the mean, standard deviation, effect size (measured by Cohen's d), and the results of the t-test for independent samples (t and p-*value*).

Social Action	Do not	Do
Favorites	37,914 (55.9%)	29,885 (44.1%)
Galleries	42,508 (62.7%)	25,291 (37.3%)
Friends	29,358 (43.3%)	38,411 (56.7%)
Comments	20,911 (30.8%)	46,888 (69.2%)

Table 5: Amount of learners who perform each of the social actions.

As shown, there is a very significant difference in favour to the social users in all the cases, both for depth (blocks), breadth (block types) and finesse (mastery). However, the biggest effect size is produced on the improvements of types of blocks and, in a slightly lesser way, on mastery; the effect size on number of blocks is much lower for every analyzed social aspect.

Another approach is to create a new variable, *sociability*, as the sum of the social actions: number of favorites, galleries, friends and comments. The median of this sociability variable is 9; we use the median to divide the set of users into *non social* (n=34,496) and *social* (n=33,303) learners. If the previous analysis is performed for these new groups, very significant differences are found for all improvements variables in favour to the *social* group, being the effect sizes $d = 0.14$ for the improvement in number of blocks, $d = 0.60$ for types of blocks, and $d = 0.50$ for mastery.

However, these differences could be attributed to the time spent in the community, as *social* users could tend to participate for longer periods of time than *non social* ones. Therefore, using the quartiles of the time variable, we divided users in four new groups, based on the number of days that they spend in the community (see Table 7).

When the improvements in learning of *social* and *non social* users are compared for each of these time groups, very significant differences are detected in favour of the *social* groups in all cases. Figures 2 and 3, which respectively illustrate the improvements in number of blocks, different types of blocks and mastery score, show that the differences in favour of the *social* group tend to increase over time.

Finally, a different approach to study the levels of social participation in the `Scratch` community can be addressed by codifying a new variable, *sociability.bis*, as the sum of each of the dichotomized social conducts. In other words, for any social conduct where a learner has shown activity, the *sociability.bis* will be increased by 1; no activity in a conduct will result in no increase. So, a learner with at least one friend and a gallery, but has not favorited and not posted comments would have a *sociability.bis* measure of 2. Five groups of users are obtained as a result as described in Table 8.

If the differences in the improvements indicators are studied in terms of level

Improvement	Group	*M*	*SD*	Cohen's *d*	*t*	*p-value*
Blocks	No favs	134.61	517.83	0.13	14.48	.000
	With favs	387.75	2,986.62			
Block types	No favs	6.73	8.53	0.56	69.13	.000
	With favs	12.66	12.75			
Mastery	No favs	1.69	2.00	0.47	58.86	.000
	With favs	2.75	2.54			
Blocks	No galleries	145.43	725.04	0.13	13.34	.000
	With galleries	415.55	3,171.00			
Blocks types	No galleries	7.06	8.91	0.58	66.52	.000
	With galleries	13.18	12.93			
Mastery	No galleries	1.76	2.05	0.48	56.85	.000
	With galleries	2.83	2.56			
Blocks	No friends	128.08	454.99	0.10	15.09	.000
	With friends	336.39	2,655.17			
Block types	No friends	6.48	8.22	0.47	63.82	.000
	With friends	11.52	12.28			
Mastery	No friends	1.63	1.94	0.41	54.19	.000
	With friends	2.56	2.49			
Blocks	No comments	91.73	349.20	0.11	19.53	.000
	With comments	315.07	2,419.73			
Block types	No comments	5.49	7.17	0.52	75.17	.000
	With comments	11.06	11.93			
Mastery	No comments	1.47	1.82	0.44	59.01	.000
	With comments	2.47	2.44			

Table 6: Differences in depth (blocks), breadth (block types) and finesse (mastery) improvements in terms of social conducts.

Number of days	Label	Number of learners
0 - 21 days	Very short time	16,956
22 - 79 days	Short time	17,068
80 - 264 days	Long time	16,829
$>$ 264 days	Very long time	16,946

Table 7: Groups of users based on the number of days that they spent in the community.

of socialization using an ANOVA analysis[3], very significant differences are de-

[3] A complete results report can be accessed in the replication package at `http://gsyc.urjc.es/~grex/repro/2016-jucs-socialization`

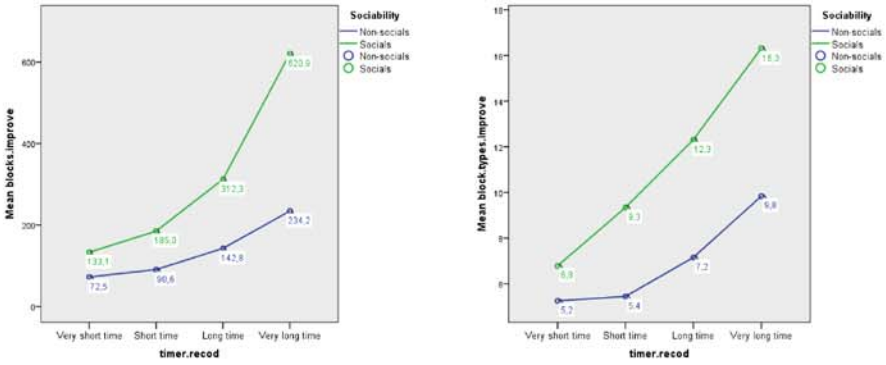

Figure 2: (l) Relationship of sociability with improvement in depth for each time group. (r) Relationship of sociability with improvement in breadth for each time group.

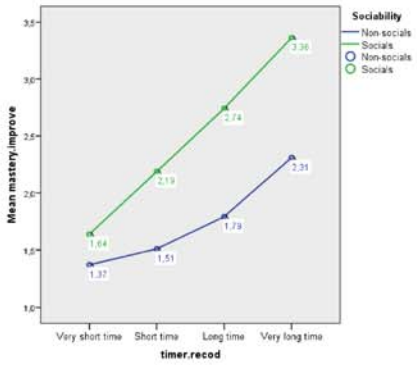

Figure 3: Relationship of sociability with improvement in finesse for each time group.

Sociability.bis	**Meaning**	**Number of learners**
0	No sociability	14,885
1	Very low sociability	11,387
2	Low sociability	11,658
3	High sociability	13,674
4	Very high sociability	16,195

Table 8: Groups of users based on *sociability.bis*, that gives the total number of social features used by a user (0 = None, 4 = All).

tected for all the cases [p(F) = .000], showing that more types of social activities in the community is positively associated with larger improvements in the programming skills. This is illustrated in Figures 4 and 5, which show the differences in improvements in depth, breadth and finesse for each social group taking the time in the community into account as a moderator variable. As can be seen, for users who spend a *long time* or a *very long time* in the community and perform the four social activities, the differences with the other groups of users in terms of improvements in number of blocks, different types of blocks and mastery score are notably bigger.

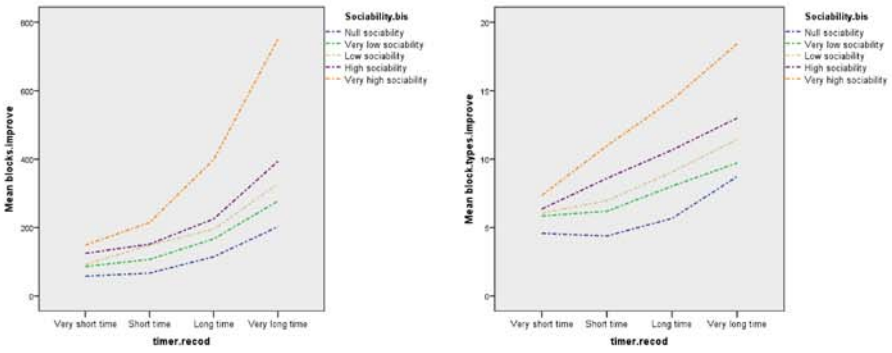

Figure 4: (l) Relationship of level of sociability with improvement in depth for each time group. (r) Relationship of level of sociability on improvement in breadth for each time group.

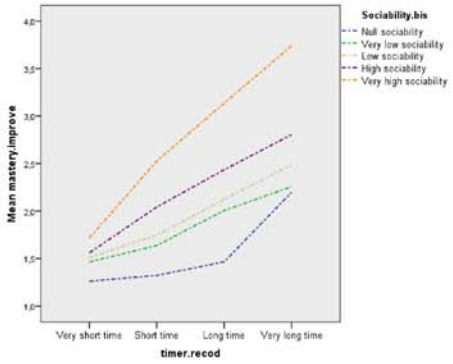

Figure 5: Relationship of level of sociability with improvement in finesse for each time group.

5.2 Relationships between remix conducts and improvements in programming skills

For the remix conducts, by dichotomizing between users who performed forks that implied modification to the source code of the original project and those who never did, two groups of users were created: *no forks* (n=20,698) and *with forks* (n=47,101). Table 9 shows the differences in improvements for both groups; there is a very significant difference in favour of the users who created forks in all cases. The biggest effect size is found for the improvements of breadth and finesse, while the effect size on depth is notably smaller.

Improvement	Group	*M*	*SD*	Cohen's *d*	*t*	*p*
Blocks	No forks	122.80	506.67	0.09	15.28	.000
	With forks	300.41	2,403.26			
Block types	No forks	6.92	8.21	0.32	44.06	.000
	With forks	10.41	11.86			
Mastery	No forks	1.75	1.96	0.26	33.25	.000
	With forks	2.34	2.43			

Table 9: Differences in depth, breadth and finesse improvements in terms of forks.

When the time spent in the community is considered as a moderator variable that affects the relationship between the independent and the dependent variables, as done in [Section 5.1], and users are therefore separated in *very short time*, *short time*, *long time* and *very long time* groups, the differences in the improvements of learning are slightly distinct. Thus, when the time spent in the community is very short, there are differences in favour of the *no forks* group when it comes to breadth and finesse. However, as time increases, the differences are in favour of the *with forks* group. Figures 6 and 7 illustrate the differences in improvements of number of blocks, different types of blocks and mastery score, and show that the relationship of remixing with the improvement of programming skills tends to increase over time.

6 Discussion

In this section we will answer the research questions that this papers targets, discuss implications and point out the threats to the validity of our results.

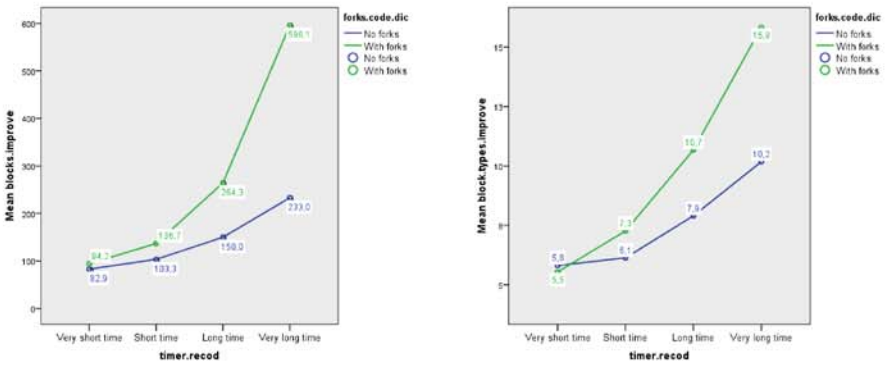

Figure 6: (l) Relationship of remix conducts with improvement in depth. (r) Relationship of remix conducts with improvement in breadth.

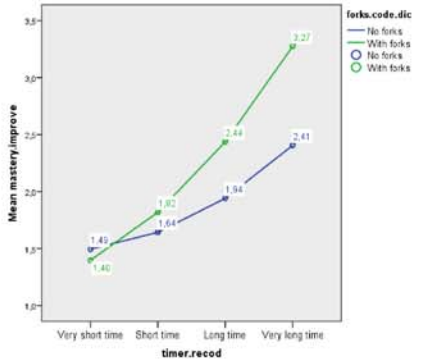

Figure 7: Relationship of remix conducts with improvement in finesse.

6.1 RQ1: Is there a relationship between learners' social participation in the Scratch community and the improvement in software development skills?

The results indicate that there is a relationship between the social conducts of learners in the community and the improvement in the sophistication of their projects, measured in depth, breadth and finesse. This is the case when any of the social activities are considered independently (*favorites*, *galleries*, *friends*, and *comments*), and also when the social conducts are grouped by the sum of the activities (codified as the *sociability* variable). In all cases there are very significant differences in favour of the *social* groups, being the effect sizes moderate (see [Ellis, 2010]) for the improvement in the amount of types of blocks and mastery score, and small for the number of blocks. Moreover, when groups of users

are created based on the level of social participation (*no sociability*, *very low sociability*, *low sociability*, *high sociability* and *very high sociability*), very significant differences are detected between groups for the improvement measurements. The results show that more social activities users perform is positively associated with more improvements in sophistication.

It must be noted that our results contradict, to some degree, the conclusions by Scaffidi et al. [Scaffidi and Chambers, 2012] and by Fields et al. [Fields et al., 2014]. On one hand, the results by Scaffidi et al., which state that a negative progression in terms of technical skills were detected for users of the `Scratch` community, are based on a sample of 250 users [Scaffidi and Chambers, 2012]. Therefore, the difference in the size of the sample can justify the discrepancies in the conclusions. In fact, Matias et al. [Matias et al., 2016] recently revisited [Scaffidi and Chambers, 2012] by studing a much bigger number of projects, and their conclusions are in line with ours, as they proved that `Scratch` users tend to share increasingly code-heavy projects over time and use a more diverse range of code concepts. In the study by Fields et al., where no link between of online participation and level of programming sophistication had been found, the sample is notably bigger, being formed by 2,225 users [Fields et al., 2014], although far from the 67,799 users analyzed in our investigation. Nonetheless, a possible explanation for the differences in the results could be based, not only on the number of users of the sample, but also on the selection process. While we selected those users who had published at least five projects during a 5 years span, Fields et al. included users who had published at least one project in a 3 months period. It is plausible that the relationships between social conducts and programming skills were not presented in their dataset since that connection emerges over time on learners with a minimum level of project production, as discussed in [Section 6.3].

6.2 RQ2: Is there a relationship between learners' remix conducts in the `Scratch` community and the improvement in software development skills?

Our results indicate that there is a relationship between the remix conducts of learners and the improvement in the sophistication of their projects, both in number of blocks, amount of types of blocks and mastery score, as very significant differences in the three cases in favour to the users who performed forks have been detected. These results are consistent with the conclusions of an investigation that also studied the whole `Scratch` repository showing that users who remix more often have larger repertoires of programming commands [Dasgupta et al., 2016]. However, the effect sizes are noticeably smaller in comparison with those of the social conducts, both for depth, breadth and finesse.

6.3 RQ3: How does time spent in the community affect the answer to questions RQ1 and RQ2?

Although results show that the relationship of social and code reuse activities with the improvement in sophistication tend to grow with time, there are slight differences among activities depending on the time the user spends in the community that are worth noting. For the social conducts, in all cases (*very short time*, *short time*, *long time* and *very long time*) there are significant differences in sophistication in favour to the *social* groups. However this is not the case for the remix conducts, as the differences are in favour of the group who does not perform forks when the time spent in the community is very short. As the time in the community increases, the differences are in favour of the group that performs remixes. Thus, the bigger differences in depth, breadth and finesse between *social* and *non social* groups, as well as between *forks* and *no forks* groups, are detected for users who spend a very long time in the community.

6.4 Implications

Taking into account that computer programming is being introduced in the national curriculum of primary and secondary education of several countries around the world [Balanskat and Engelhardt, 2015], these findings, if confirmed in controlled studies, could have an impact on the teaching of software development skills in schools, high schools and universities alike. Educators could discriminate between programming platforms in terms of the social, remix, and software evolution features that are offered, and could also plan the right time to encourage learners to put into play the different social and remix features provided by the programming platform.

But not only the education sector could exploit these findings. Companies and organizations building different types of tools to support programming activities, both focused on learners and professional developers, could also take them into account to include new and diverse social functionalities, which could have an impact on the progression of the developers using the platform. In a similar vein, both individual programmers and software development teams could try to encourage social participation as a mean to enhance learning.

Our results, which show a positive correlation between the development of software programming skills and the amount of social conducts of the learner in the `Scratch` platform, may contradict a widespread idea (almost a stereotype): excellent programmers do not like social interaction [Schott and Selwyn, 2000]. In fact, we have found some literature, prior to the year 2000, which supported the idea of the tendency of programmers to introversion. So, in [Ketler and Smith, 1992] programmers were characterized as more introverted than the general population; [Kagan, 1984] reported that extroversion is negatively correlated with

achievement in computer programming exams; and in [Bush and Schkade, 1985] most programmers were identified as being introverted, serious, quiet and logical. However, some other studies did not find relationship between introversion/extroversion and computer programming [Kagan and Douthat, 1985, Newsted, 1975].

A second body of research, especially from the year 2008 onwards and coinciding with the raise of social software development platforms (such as GitHub or GitLab) and social coding site for learning to program (such as `Scratch`), started to find evidence of the contrary: the superiority of extroverts in programming task. Hence, in [Law et al., 2010] extrovert students outperformed introvert students when learning Java; results in [Yilmaz and O'Connor, 2012] indicate that there are more individuals in software teams, who may perceive to be extroverted not only in a classroom environment but also in an industrial setting. Moreover, in [Licorish and MacDonell, 2014] top global software practitioners demonstrated more openness to experience than the other practitioners did; additionally, top practitioners involved in usability-related tasks were found to be highly extroverted. Even more, Bazelli et al. analyze the Q&A website StackOverflow for personality traits of participants [Bazelli et al., 2013]: using textual analysis of posts, they found that authors with higher reputation are more extroverted compared to those with medium and lower reputation. Finally, a recent research has found a positive correlation between computational thinking, which is supposed to be the problem-solving ability that underlies programming tasks, and the extroversion personality factor in a sample of Spanish middle-school students [Román-González et al., 2016b]. Our results are consistent with this emerging second body of research.

Moving the discussion to educational environments for learning programming, it is plausible that the first wave of programming languages in schools (e.g., `Logo`) could fit better to introverted learners, while the second wave with visual programming languages (e.g., `Scratch`) is better suited to the extroverted ones. It should be noted that the former had difficult syntaxes, non-significant products and were essentially lonely-learning environments, while the latter overcome the aforementioned limitations providing an easy floor with easy syntax, wide walls due to diverse and significant coding products, and social learning environment that allows remixing and high ceiling products. Maybe we are witnessing a transformation of the programmer from a *logical-formal*, *inside-oriented* to an *expressive-communicative*, *outside-oriented* subject. This path was already anticipated by Capretz in 2003 (between the two waves) when warning that if the software field continued dominated by introverted it would be difficult to meet the user requirements and the social needs; so more extroverted were desired [Capretz, 2003].

6.5 Threats to validity

There are several factors that affect the validity of our investigation. On the one hand, as this dataset does not offer access to the .sb2 files with the actual source code of the projects developed by learners, there is no possibility of measuring how effectively blocks are utilized in the projects. We can count the number of blocks, add the number of different types of blocks, and check whether learners use advanced and complex instructions, but there is no evidence that those blocks are used correctly.

Furthermore, our analysis is based on the notion of sophistication of programming skills, which is based on breadth, depth and finesse, may not be ideal to measure software development learning. Other approaches based, for instance, on bugs or on reusability of the code, could yield different results.

Moreover, there is no indication to state that users have learned in similar circumstances. It might be possible that *social* users have learned in formal environments, with help from peers and teachers, and being encouraged to share and participate in the community. On the contrary, *non social* users might mainly have learned in informal settings by their own. Although the size of the sample mitigates this potential threat to validity, there is no information in the dataset about the environment where users developed their programming learning.

It must be taken into account that even though we used the terms dependent, independent and moderator variables along the text, this work is not an experiment, but a posterior investigation of social interactions and products created by a community of learners. Consequently, the relationships between the so-called dependent and independent variables cannot be understood as causality. Future research, as detailed in [Section 7], should validate our results and state stronger conclusions by means of a controlled study.

In addition to dividing learners into quartiles (i.e., forming groups of same size), we have divided learners into equal timeframes (in our case, we chose 80 days) to address the concern that from a statistical perspective the chosen day divisions may have served to support the research hypotheses, even if inadvertently. The results using this second method are consistent with those presented in the paper, since social users have a greater improvement than non-social users in all groups. The specific results for this analysis are publicly available in the replication package.

Finally, when it comes to generalization to other, professional environments, these results must be treated with caution, being our investigation based on a learning community where most of the learners are non-professional developers under 18 years old. In this regard, although no identifiable information about users, including age, is incorporated in the dataset, stats from the `Scratch` website[4] indicate that a vast majority of users are under 18. It should be noted as

[4] See https://scratch.mit.edu/statistics/

well that there is a difference between learning to code and professional coding, where developers predominately already know how to code. Consequently, observations from a code learning environment such as `Scratch` may not transfer directly to professional software development environments.

7 Conclusions and Future Work

This paper presents an investigation of the social and remix conducts of almost 70,000 users and the sophistication of over 1.5 million software projects authored by them in the `Scratch` platform, the most popular social coding site for learning to program. The results indicate that there is a relationship between the social conducts of users and the improvement of their programming abilities, showing that more social actions performed by users is positively associated with more sophistication in their programs. The relationship between remix conducts and coding skills is also detected, although the effect sizes on the improvement are noticeably smaller than in the former case. Furthermore, the results also provide evidence that the relationship of both social and remix factors with the development of software programming skills tends to grow with time.

It must be noted, though, that this work was not an experiment, but a posterior investigation. Therefore, the relationships between the social and remix conducts with software development skills cannot be considered as causality. In the near future we plan to carry out a controlled experiment with two groups of learners formed by students with similar characteristics regarding to age, gender and computational thinking skills, measured using the `CT-test` [Román-González et al., 2016a]. One of the groups will learn software development using the offline `Scratch` version, while the other one will make use of social features offered in the online community.

Since 2012, with the release the new `Scratch` site, important changes with the aim of boosting participation of learners in the community have been introduced. Access to the new dataset would allow to further study the differences in the participation patterns and in the learning of software development skills.

Further research on the topic should address the causes of the increased socialization and the corresponding improvement in programming outcomes. Other factors such as aptitude, enjoyment (and sharing) of programming tasks or actual time devoted to programming could be reasonable causes of this behavior. Therefore, deeper discussion, including other research approaches, should be desirable.

Finally, further investigations should replicate this investigation using data from a professional social coding site, such as GitHub, which could allow to study differences in the impact of social activities on software development skills between mostly young learners and professional developers.

Acknowledgments

The work of the authors has been funded in part by the Madrid Region under eMadrid (S2013/ICE-2715) and the second author by the Spanish Gov. with SobreSale (TIN2011-28110). We are thankful to the `Scratch` Research Data *team* for granting us access to the data, especially to Benjamin Mako Hill and Andrés Monroy-Hernández.

References

[Ahmadi et al., 2008] Ahmadi, N., Jazayeri, M., Lelli, F., and Nesic, S. (2008). A survey of social software engineering. In *Automated Software Engineering-Workshops, 2008. ASE Workshops 2008. 23rd IEEE/ACM International Conference on*, pages 1–12. IEEE.

[Augustine et al., 2005] Augustine, S., Payne, B., Sencindiver, F., and Woodcock, S. (2005). Agile project management: Steering from the edges. *Commun. ACM*, 48(12):85–89.

[Babb et al., 2014] Babb, J., Hoda, R., and Norbjerg, J. (2014). Embedding reflection and learning into agile software development. *IEEE Software*, 31(4):51–57.

[Balanskat and Engelhardt, 2015] Balanskat, A. and Engelhardt, K. (2015). Computing our future: Computer programming and coding - priorities, school curricula and initiatives across Europe. Technical report, European Schoolnet.

[Bazelli et al., 2013] Bazelli, B., Hindle, A., and Stroulia, E. (2013). On the personality traits of stackoverflow users. In *ICSM*, pages 460–463.

[Boe et al., 2013] Boe, B., Hill, C., Len, M., Dreschler, G., Conrad, P., and Franklin, D. (2013). Hairball: Lint-inspired static analysis of Scratch projects. In *44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 215–220, NY, USA. ACM.

[Bougie et al., 2011] Bougie, G., Starke, J., Storey, M.-A., and German, D. M. (2011). Towards understanding Twitter use in Software Engineering: Preliminary findings, ongoing challenges and future questions. In *Proceedings of the 2nd international workshop on Web 2.0 for software engineering*, pages 31–36. ACM.

[Brennan et al., 2010] Brennan, K., Monroy-Hernández, A., and Resnick, M. (2010). Making projects, making friends: Online community as catalyst for interactive media creation. *New Directions for Youth Development*, 2010(128):75–83.

[Bush and Schkade, 1985] Bush, C. M. and Schkade, L. L. (1985). In search of the perfect programmer. *Datamation*, 31(6):128–132.

[Capiluppi et al., 2013] Capiluppi, A., Serebrenik, A., and Singer, L. (2013). Assessing technical candidates on the social web. *Software, IEEE*, 30(1):45–51.

[Capretz, 2003] Capretz, L. F. (2003). Personality types in software engineering. *International Journal of Human-Computer Studies*, 58(2):207–214.

[Clarke et al., 2015] Clarke, P., O'Connor, R. V., Leavy, B., and Yilmaz, M. (2015). Exploring the relationship between software process adaptive capability and organisational performance. *IEEE Transactions on Software Engineering*, 41(12):1169–1183.

[Dabbish et al., 2012] Dabbish, L., Stuart, C., Tsay, J., and Herbsleb, J. (2012). Social coding in GitHub: Transparency and collaboration in an open software repository. In *ACM 2012 conference on Computer Supported Cooperative Work*, pages 1277–1286. ACM.

[Dahotre et al., 2010] Dahotre, A., Zhang, Y., and Scaffidi, C. (2010). A qualitative study of animation programming in the wild. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, page 29. ACM.

[Dasgupta et al., 2016] Dasgupta, S., Hale, W., Monroy-Hernández, A., and Hill, B. M. (2016). Remixing as a pathway to computational thinking. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*, CSCW '16, pages 1438–1449, New York, NY, USA. ACM.

[Ellis, 2010] Ellis, P. D. (2010). *The essential guide to effect sizes: Statistical power, meta-analysis, and the interpretation of research results*. Cambridge University Press.

[Fields et al., 2014] Fields, D. A., Giang, M., and Kafai, Y. (2014). Programming in the wild: Trends in youth computational participation in the online Scratch community. In *9th workshop in primary and secondary computing education*, pages 2–11. ACM.

[Fields et al., 2013] Fields, D. A., Giang, M., and Kafai, Y. B. (2013). Understanding collaborative practices in the Scratch online community: Patterns of participation among youth designers. In *To see the world and a grain of sand: Learning across levels of space, time, and scale: CSCL 2013 Conference Proceedings*, volume 1, pages 200–207. International Society of the Learning Sciences.

[Gross et al., 2010] Gross, P. A., Herstand, M. S., Hodges, J. W., and Kelleher, C. L. (2010). A code reuse interface for non-programmer middle school students. In *Proceedings of the 15th international conference on Intelligent user interfaces*, pages 219–228. ACM.

[Haefliger et al., 2008] Haefliger, S., Von Krogh, G., and Spaeth, S. (2008). Code reuse in open source software. *Management Science*, 54(1):180–193.

[Halstead, 1977] Halstead, M. H. (1977). *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Inc.

[Hoda et al., 2013] Hoda, R., Babb, J., and Norbjerg, J. (2013). Toward learning teams. *Software, IEEE*, 30(4):95–98.

[Hoda et al., 2011] Hoda, R., Noble, J., and Marshall, S. (2011). Developing a grounded theory to explain the practices of self-organizing agile teams. *Empirical Software Engineering*, 17(6):609–639.

[Huff et al., 1992] Huff, S. L., Munro, M. C., and Marcolin, B. (1992). Modelling and measuring end user sophistication. In *Proceedings of the 1992 ACM SIGCPR Conference on Computer Personnel Research*, SIGCPR '92, pages 1–10, New York, NY, USA. ACM.

[Jiang et al., 2013] Jiang, J., Zhang, L., and Li, L. (2013). Understanding project dissemination on a social coding site. In *2013 20th Working Conference on Reverse Engineering (WCRE)*, pages 132–141. IEEE.

[Kagan, 1984] Kagan, D. M. (1984). Personality and learning basic. *Journal of Instructional Psychology*, 11(1):10.

[Kagan and Douthat, 1985] Kagan, D. M. and Douthat, J. M. (1985). Personality and learning fortran. *International journal of man-machine studies*, 22(4):395–402.

[Ketler and Smith, 1992] Ketler, K. and Smith, R. D. (1992). Differences between third and fourth generation programmers: A human factor analysis. *Information Resources Management Journal (IRMJ)*, 5(2):25–35.

[Law et al., 2010] Law, K. M., Lee, V. C., and Yu, Y.-T. (2010). Learning motivation in e-learning facilitated computer programming courses. *Computers & Education*, 55(1):218–228.

[Licorish and MacDonell, 2014] Licorish, S. A. and MacDonell, S. G. (2014). Personality profiles of global software developers. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, page 45. ACM.

[Matias et al., 2016] Matias, J. N., Dasgupta, S., and Hill, B. M. (2016). Skill progression in scratch revisited. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 1486–1490. ACM.

[McCabe, 1976] McCabe, T. (1976). A complexity measure. *Software Engineering, IEEE Transactions on*, SE-2(4):308–320.

[Mens and Goeminne, 2011] Mens, T. and Goeminne, M. (2011). Analysing the evolution of social aspects of open source software ecosystems. In *IWSECO@ ICSOB*,

pages 1–14.

[Monroy-Hernández and Resnick, 2008] Monroy-Hernández, A. and Resnick, M. (2008). Empowering kids to create and share programmable media. interactions, 15: 50–53. *ACM ID*, 1340974.

[Moreno-León et al., 2015] Moreno-León, J., Robles, G., and Román-González, M. (2015). Dr. Scratch: Automatic analysis of Scratch projects to assess and foster computational thinking. *RED. Revista de Educación a Distancia*, 15(46).

[Moreno-León et al., 2016a] Moreno-León, J., Robles, G., and Román-González, M. (2016a). Comparing computational thinking development assessment scores with software complexity metrics. In *Global Engineering Education Conf. (EDUCON), 2016 IEEE*, pages 1040–1045.

[Moreno-León et al., 2016b] Moreno-León, J., Robles, G., and Román-González, M. (2016b). How social are Scratch learners? A comprehensive analysis of the Scratch platform for social interactions. In *FLOSS Education and Computational Thinking Workshop. 12th International Conference on Open Source Systems*, pages 19–26.

[Newsted, 1975] Newsted, P. R. (1975). Grade and ability predictions in an introductory programming course. *ACM SIGCSE Bulletin*, 7(2):87–91.

[Pham et al., 2013] Pham, R., Singer, L., Liskin, O., Figueira Filho, F., and Schneider, K. (2013). Creating a shared understanding of testing culture on a social coding site. In *Software Engineering (ICSE), 2013 35th Intl Conference on*, pages 112–121. IEEE.

[Resnick, 2012] Resnick, M. (2012). Mother's day, warrior cats, and digital fluency: Stories from the Scratch online community. In *Proceedings of the Constructionism 2012 Conference: Theory, Practice and Impact*, pages 52–58.

[Resnick et al., 2009] Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., and Kafai, Y. (2009). Scratch: Programming for all. *Commun. ACM*, 52(11):60–67.

[Román-González et al., 2016a] Román-González, M., Pérez-González, J.-C., and Jiménez-Fernández, C. (2016a). Which cognitive abilities underlie computational thinking? criterion validity of the computational thinking test. *Computers in Human Behavior*.

[Román-González et al., 2016b] Román-González, M., Pérez-González, J. C., Moreno-León, J., and Robles, G. (2016b). Does computational thinking correlate with personality? the non-cognitive side of computational thinking. In *Proceedings of the Technological Ecosystems for Enhancing Multiculturality Conference, TEEM '16*, pages 1–10.

[Sawyer and Guinan, 1998] Sawyer, S. and Guinan, P. J. (1998). Software development: Processes and performance. *IBM Systems Journal*, 37(4):552–569.

[Scaffidi and Chambers, 2012] Scaffidi, C. and Chambers, C. (2012). Skill progression demonstrated by users in the Scratch animation environment. *International Journal of Human-Computer Interaction*, 28(6):383–398.

[Schott and Selwyn, 2000] Schott, G. and Selwyn, N. (2000). Examining the male, antisocial stereotype of high computer users. *Journal of Educational Computing Research*, 23(3):291–303.

[Storey et al., 2010] Storey, M.-A., Treude, C., van Deursen, A., and Cheng, L.-T. (2010). The impact of social media on software engineering practices and tools. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pages 359–364. ACM.

[Surian et al., 2010] Surian, D., Lo, D., and Lim, E.-P. (2010). Mining collaboration patterns from a large developer network. In *Reverse Engineering (WCRE), 2010 17th Working Conference on*, pages 269–273. IEEE.

[Thung et al., 2013] Thung, F., Bissyandé, T. F., Lo, D., and Jiang, L. (2013). Network structure of social coding in GitHub. In *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*, pages 323–326. IEEE.

[Vasilescu et al., 2013] Vasilescu, B., Filkov, V., and Serebrenik, A. (2013). StackOverflow and GitHub: Associations between software development and crowdsourced

knowledge. In *Social Computing (SocialCom), 2013 International Conference on*, pages 188–195. IEEE.

[Velasquez et al., 2014] Velasquez, N. F., Fields, D. A., Olsen, D., Martin, T., Shepherd, M. C., Strommer, A., and Kafai, Y. B. (2014). Novice programmers talking about projects: What automated text analysis reveals about online scratch users' comments. In *System Sciences (HICSS), 2014 47th Hawaii Intl Conference on*, pages 1635–1644. IEEE.

[Wolz et al., 2011] Wolz, U., Hallberg, C., and Taylor, B. (2011). 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*.

[Yilmaz and O'Connor, 2012] Yilmaz, M. and O'Connor, R. V. (2012). Towards the understanding and classification of the personality traits of software development practitioners: Situational context cards approach. In *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*, pages 400–405. IEEE.

[Zuckerman et al., 2009] Zuckerman, O., Blau, I., and Monroy-Hernández, A. (2009). Children's participation patterns in online communities: An analysis of Israeli learners in the Scratch online community. *Interdisciplinary Journal of E-Learning and Learning Objects*, 5(1):263–274.