Skill progression demonstrated by users in the Scratch animation environment

Christopher Scaffidi, Christopher Chambers
{ cscaffid , chambech }@eecs.oregonstate.edu

Corresponding Author
Christopher Scaffidi
cscaffid@eecs.oregonstate.edu
School of Electrical Engineering and Computer Science, Oregon State University
1148 Kelley Engineering Center, Oregon State University, Corvallis, OR 97331-4501
541-737-5572 (phone)
360-935-7708 (fax)

Abstract

The Scratch environment exemplifies a tool+community approach to teaching elementary programming skills, as it includes a website where users can publish, discuss, and organize animations that are programs. To explore this environment's effectiveness for helping people to develop programming skills, we performed a quantitative analysis of 250 randomly-selected users' data, including over 1000 of their animations. We measured skill based on four models that had proven useful in prior empirical studies. Overall, we found mixed results about the environment's effectiveness. Among users who do not drop out, we found an increasing progression in social skills. However, we also observed an extremely high drop-out rate. Moreover, we observed a flat or decreasing level of demonstrated skill on virtually every measure. These results call into question whether simply combining an animation tool and an online community is sufficient for keeping people engaged long enough to learn elementary programming skills.

Keywords: Empirical studies; Novice animation programmers

1 Introduction

Animation programming has played a crucial historical role in motivating and helping people to learn the elementary basics of programming such how to code loops, conditionals, and event handlers. Over the past fifty years, numerous animation programming environments such as Logo (Papert, 1980), KidSim (Cypher et al., 2001), AgentSheets (Repenning, 1993), Alice (Cooper et al., 2000) and Hands (Pane, 2002) have appeared, each offering new features for getting students interested in learning elementary programming skills, for easing the difficulty of creating animations, and for creating specific new kinds of animations not previously supported. The goal with these tools is not to turn every user into a professional programmer who creates programs, but rather to help people develop elementary programming skills that they can apply later in everyday work and life.

Representing one of the most recent iterations in this series of programming tools, the Scratch environment includes a website where users can publish animations and comment upon one another's work (Resnick *et al.*, 2009). We refer to this as a "tool+community" approach, since the website provides learners with a vast supply of existing animations as resources for learning elementary programming skills. The approach also offers an opportunity for people to learn *social* skills related to programming, such as the ability to share and contribute to communal resources. Scratch was originally introduced through afterschool clubs (Monroy-Hernández and Resnick, 2008), but recently it has been more widely adopted by users on the web at large (Resnick *et al.*, 2009), where its registered user base now exceeds 800,000.[1] Scratch was primarily designed to encourage young people to program, and the majority of these users are novices with little previous programming experience (Resnick *et al.*, 2009).

Previous studies have suggested that Scratch is useful for exercising elementary programming skills. In one study, Scratch's designers analyzed 536 animations created in the afterschool club; they found that most programming primitives such as loops, conditionals and event handlers were widely used by students at the club, and they found that some primitives were more heavily used during Year 2 of the club than in Year 1 (Maloney *et al.,* 2008). In our own research group's prior work, we analyzed 100 animations downloaded from the website and found that, as in the club setting, most programming primitives were also widely used by Scratch

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

users on the web (Dahotre *et al.*, 2010). From these studies, we conclude that Scratch animations demonstrate effective use of elementary programming skills, both within a club and on the web.

What these studies have not explored is whether the Scratch environment is useful for *developing* social skills or elementary programming skills over the web. In previous work, we only looked at a single snapshot of animations on the website (Dahotre *et al.*, 2010), so we do not know whether users demonstrate *increasing* skill over time. In the afterschool club setting, users had mentors, who would periodically teach new concepts. Such a mentor can be indispensible for learning. For example, "one user [in the club] had a desperate need for the variables in his project. When Mitchel Resnick, on a visit to the Clubhouse, showed him how to use variables, he immediately saw how they could be used to solve his problems and thanked Mitchel repeatedly for the advice" (Maloney *et al.*, 2008). On the web, novice Scratch users have no skilled mentor standing by their side. The only source of teaching is the community and its resources, but the community's members might not be much more skilled than any particular user. Given the differences between the web and the club setting, it is therefore unclear whether Scratch is indeed an effective environment for developing social skills or elementary programming skills.

Thus, the question remains: *To what extent do users in the Scratch environment demonstrate increasing skills over time?* This question has significant implications for the development and deployment of online environments aimed at teaching these basic skills. The reason is that Scratch is a widely-used, well-respected environment that has shown excellent promise in the club setting. Therefore, it represents something of a "best case" for what we can expect novices to learn through the tool+community approach for designing animation environments. If Scratch users are *not* learning much from this environment, then one must question whether the simple approach of combining an animation tool with an online community is really enough to stimulate learning of elementary programming skills. On the other hand, if Scratch is indeed succeeding, then we can infer that it offers a viable prototype for teaching novice users these skills and perhaps other more advanced animation-programming skills. For example, future work could explore the extent to which Scratch enables users to learn how to code common temporal behaviors or how to correctly use concepts related to geometric constraints in their animations.

In order to answer our primary question above, we analyzed 250 randomly-selected Scratch users' data, including over 1000 of their Scratch animations. To frame this analysis, we adapted four existing models of end users. One of these models describes progression of social skills, while another two describe different aspects of elementary programming skill related to usage of programming primitives. The fourth model brings in a time element and explores whether users are increasing in efficiency as measured by amount of code written per unit time. These models had proven useful in prior empirical studies; however, they had not been used previously in the context of programming in an animation environment such as Scratch. Therefore, in addition to our primary contribution of answering our central research question above, our secondary contribution is in showing how these models can be adapted in order to investigate skill progression in Scratch, which exemplifies the tool+community approach to designing animation environments.

Overall, we have found mixed results about the effectiveness of the Scratch environment. Among users who do not drop out of the community, we found a progression in skills related to social interaction. On the other hand, we also observed an extremely high rate of drop-out among Scratch users, with only 3 months being the median duration of involvement in the online Scratch environment. Worse, Scratch animations published over time actually demonstrated a flat or *decreasing* level of skill on virtually every single measure, rather than a steady increase as would be hoped. In particular, the numbers and varieties of programming primitives used in Scratch animations showed a broad and statistically significant drop over time, both during the first few months and during longer time intervals, and the speed with which people created these primitives was essentially flat over time (or slightly worsening). These disappointing results have several possible explanations, one of which is that Scratch's tool+community approach might not suffice for keeping users engaged long enough to learn elementary programming skills.

The remainder of this paper is organized as follows. In Section 2, we review background and related work that highlights the crucial historical role that Scratch and other animation tools have played in teaching elementary programming skills within classroom settings. In Section 3, we summarize the models that have guided our analysis, including a discussion of how we adapted these models to frame our study. In Section 4, we describe the data that we acquired, while in Section 5, we present our analysis and results. In Section 6, we discuss interpretations, implications, limitations, and opportunities for future work.

2 Background and related work

In Scratch and similar environments for creating animations, each animation is a program. For Scratch in particular, each animation consists of a set of animated sprites, which are images whose position and appearance are controlled with programmatic scripts (Figure 1) (Resnick *et al.*, 2009). When some users start using Scratch, they begin by creating static images, which does not require creating any scripts (Resnick *et al.*, 2009). However, it is impossible to create *animated* images in Scratch without writing programmatic scripts. For example, it is impossible to make an animation respond to mouse input without coding an event handler, and it is impossible to create ongoing progressive behavior (such as continually rotating an image) without coding a loop. The creation of an "animated animation" in Scratch is, therefore, a programming activity.

Perhaps the earliest such sprite-based animation environment was Logo, a textual language and supporting toolset introduced in the late 1960's (Papert, 1980). More recent environments such as Alice have added sophisticated new features, such as support for 3D animation (Cooper *et al.*, 2000). Other tools such as AgentSheets (Repenning, 1993), KidSim (Cypher *et al.*, 2001) and Hands (Pane, 2002) have augmented the textual scripting language with programming-by-demonstration (PBD), a feature that enables users to specify example inputs and outputs, from which the tool infers customizable scripts for sprites. (Scratch lacks PBD.) Laboratory experiments and field tests in classrooms have shown that animation tools can serve as effective platforms for teaching use of programming primitives (Maloney *et al.*, 2008; Pane, 2002), algorithm design (Cooper *et al.*, 2000) and problem-solving skills (Pane, 2002; Repenning, 1993; Koh *et al.*, 2010), as well as for motivating children and computer science majors to continue programming after they leave the classroom (Cypher *et al.*, 2001; Moskal *et al.*, 2004).

While a few previous environments included a website where users could upload their animations (e.g., Cooper *et al.*, 2000; Stahl *et al.*, 1995), Scratch has greatly scaled up this tool+community approach: whereas earlier environments encompassed only a few thousand users, Scratch's registered user base is now over 800,000. The goal of providing this integrated tool+community approach is to help users to learn social skills in addition to elementary technical programming skills (Monroy-Hernández and Resnick, 2008; Resnick *et al.*, 2009). The provision of an online community has previously been found to assist in the development of social skills by people in a wide range disciplines, such as by novice psychologists (Mebane *et al.*, 2008), teenagers learning to do marketing (Wang *et al.*, 2001), and even professional software engineers (Hofmann and Wulf, 2003). It is therefore reasonable to expect that augmenting an animation tool that is useful for teaching elementary programming skills might thereby enable users also to learn relevant social skills.

For example, one important way in which a Scratch user can practice social skills is by setting up a "gallery," which is an area of the website created by users for the purpose of eliciting contributions from other people. Galleries serve as social hubs that give structure to the community. In particular, galleries can become a nexus for meeting, "friending" (where users indicate that they share a social bond), and publicizing joint accomplishments (Monroy-Hernández and Resnick, 2008; Resnick *et al.*, 2009). Outside of galleries, users can post comments on each others' animations and exchange tips with one another via threaded forums. They also can download and adapt ("remix") one another's code.

Scratch's designers have conducted substantial research to evaluate the effectiveness of the environment. In one study, they deployed the programming tool to teenagers in an afterschool club for two years. To track development of technical skill, they measured how much various programming primitives such as loops and variables were used by users during the first and second years. This study revealed that most primitives' usage showed a small but statistically significant increase between the two years (Monroy-Hernández and Resnick, 2008). As evidence of Scratch's effectiveness for helping people to exercise social skills related to programming, the environment's designers have harvested many anecdotes from the online community since its launch in May 2007 (Monroy-Hernández and Resnick, 2008; Resnick *et al.*, 2009). For example, one anecdote tells of two girls who created "a miniature company" intended to publish "top quality games" created with Scratch. They set up a gallery and posted some animations. Soon, two boys discovered the gallery and offered to join the company (Resnick, 2007). Together, these pieces of research suggest that the programming tool is an effective platform for exercising technical skills in the club setting and that adding a community enables at least some users to exercise social skills related to collaborative programming.

In order to assess whether these anecdotes are consistent with the *typical* user's experience on the website, our previous work analyzed whether 100 randomly-selected animations demonstrated the same level of elementary technical programming skill as had been reported in the clubhouse setting. We found, indeed, that online animations demonstrated at least as much usage of each programming primitive as had been previously reported for the clubhouse (Dahotre *et al.*, 2010). Moreover, by manually reviewing every animation in our sample, we concluded that primitives were almost always used in a way that appeared perfectly correct—in fact, only 7 of the 100 animations had any obvious bugs at all. In an informal qualitative investigation, we found that approximately 10% of animations revealed evidence of higher-level design. In addition, we also investigated whether any of these animations were coded collaboratively, but we found no evidence that they had been. A qualitative analysis of users' online comments to one another indicated, however, that most discussions were collegial and often aimed at sharing constructive feedback, although we also observed that 26% of comments were about topics unrelated to Scratch (e.g., music and video games). From these studies, we concluded that the tool+community approach helps users to exercise technical skills and some collaborative social skills.

One major limitation of our prior work is that we examined a *snapshot* of animations, comments, and forum threads, so we could not conclude that users in the online community were *learning* anything over time. They may instead have merely been exercising skills that they already possessed when they started using Scratch. Even if students were learning elementary technical programming skills over time in the club setting, the same need not necessarily be true online, where no mentor is present. Therefore, we have conducted a new study to investigate whether Scratch users demonstrate a *progression* of social and elementary technical programming skills over time, in order to evaluate how well the tool+community approach supports learning.

3 Models of user progression

To frame our analysis of Scratch users, we adapted four models of programmers that have proven useful in prior empirical studies. This adaptation has yielded several quantitative measures of demonstrated skill, thereby reducing the need for qualitative methods (such as those used in our prior work) and facilitating the use of statistical tests for determining whether Scratch users demonstrate increasing levels of skill.

3.1 Onion model: Progression by kind and amount of activity

Based on an analysis of human behavior in communities that produce open source software, such as the PostgreSQL database server, the following 8-category taxonomy has been proposed for classifying people on a range from user up through developer and project leader (Nakakoji *et al.*, 2002; Ye and Kishida, 2003):

- Passive Users: people who just use the software (e.g., PostgreSQL)
- Readers: people who try to understand the software's source code
- Bug Reporters: people who discover and report defects in the software
- Bug Fixers: people who modify the software to correct defects
- Peripheral Developers: people who occasionally fix bugs or implement new features
- Active Developers: people who regularly fix bugs or implement new features
- Core Members: people who guide and coordinate projects
- Project Leaders: people who coordinate and set vision for projects that they initiate

The taxonomy describes a process whereby a person starts with simple use, moves to shared ownership, and then continues on to leadership. At each stage, the person has "a larger radius of influence" (Ye and Kishida, 2003), meaning that their actions affect more people. Mastery of levels, and therefore progression, is characterized as development of social skills situated within a community of practice. The model is often represented with an onion-like diagram (Figure 2), where circles represent sets and people move toward the smaller, more socially adept subsets over time. Empirically, people may spend months to years at each level, and some never move beyond the outermost levels. Overall, for open source communities to be viable, they "have to be able to regenerate themselves through the contributions of their members and the emergence of new Core Members and Active Developers" (Ye and Kishida, 2003)—without social progression of individuals, these communities die.

Adapting the model to Scratch

We used the onion model as a starting point for developing measures of social skill in the Scratch environment. Although MIT owns and controls the actual Scratch programming tool, the website is part of the environment experienced by Scratch users, and members of the community can contribute to the environment in three ways. First, they can use the environment to create and share animations. Second, although bug fixing is rare (since bugs are rare, as explained above), remixing does sometimes occur and is largely aimed at program enhancement. Third, galleries play a role similar to open source projects by providing a structure to frame contributions; thus, we anticipated that contribution to galleries, and creation of galleries, might be analogous to the leadership activities of open source developers in the highest categories of the onion model.

In order to test whether these three forms of contribution by novice Scratch animation users reveal a progression similar to that of professional developers in traditional open source projects, we adapt the onion model as follows:

```
if (user has created ≥ 25 non-empty Scratch galleries) then
  category = “Project Leader”
else if (user contributes to ≥ 15 non-empty galleries) then
  category = “Active User”
else if (user creates ≥ 10 animations) then
  category = “Peripheral User”
else if (user creates ≥ 5 animations and ≥ $^1/_3$ of those are remixes) then
  category = “Remixer”
else
  category = “Passive User”
```

In our adaptation of this model, we omit three onion-model categories for which we cannot easily identify members, given the data available on the Scratch website (i.e., Readers, Bug Reporters, and Core Members). In addition, we prefer to refer to “Bug Fixers” as “Remixers,” since only a third of remixes are typically bug fixes (Dahotre *et al.*, 2010). Finally, we have selected numerical thresholds for categorizing Scratch users because prior work did not specify what thresholds to use (Ye and Kishida, 2003) and because some thresholds are necessary in order to obtain quantitative measures. (We have also tried using our adapted model with other thresholds and found no interesting differences that depended on the specific choice of threshold.)

3.2 Sophistication: Progression in breadth, depth and finesse

In addition to the social viewpoint offered by the onion model, we needed measures of elementary technical programming skill for our study. After investigating the literature, we found that nearly two decades ago, Huff et al. (1992) developed a survey that could be used to classify end users based on programming skills. The survey asked people whether they regularly used specific programming features, such as data-manipulation features in spreadsheets and Unix commands in shell scripts. From these data, three measures of skill were derived. The first measure, “breadth,” referred to the range of different features that people could use. The second measure, “depth,” referred to the amount with which people used these features. The third measure, “finesse,” referred to a user’s ability to solve programming problems effectively and creatively. The three forms of skill measured by this survey were together referred to as “sophistication” (Huff *et al.*, 1992) (later renamed “competence” (Munro *et al.*, 1997)). Empirically, the three dimensions are highly correlated, which “indicates that, on average, users tend to develop their breadth and depth of knowledge, and their finesse in applying IT, in parallel (though any particular user may be noticeably ‘unbalanced’ in term [sic] of the three dimensions)” (Huff *et al.*, 1992). While we are unsure whether “unbalanced” is exactly the right word for people who lack an even distribution of ability across the three dimensions, we accept that all three dimensions of this model represent useful aspects of elementary programming skill, so we have adapted this model for use in our analysis.

Breadth and depth in the context of Scratch

Scratch offers approximately 120 different programming primitives that we have grouped into 17 categories (Table 1). By looking at animation code from the website, we can determine which primitives are invoked by an animation and how often. Thus, we can directly compute the total *number* of primitives invoked in an animation as a measure of depth, and we can compute the total number of *distinct* categories of primitives used per animation as a measure of breadth. This eliminates the need to measure skill through a survey, which would rely on users’ recollection and self-perception.

Finesse in the context of Scratch

The model’s third measure of sophistication, finesse, refers to a user’s ability to solve programming problems effectively and creatively. Different programming environments present substantially different problems and require different kinds of solutions, so the concept of finesse must be contextualized to programming in the Scratch environment. Therefore, we performed an informal review of animations on the Scratch website to identify several key problems and solutions, from which we created measures for characterizing finesse. Our informal review revealed four problems and related solutions where we judged that some animations showed more effectiveness and creativity than others.

First, we observed that some sprites had visual asymmetry such that their orientation was semantically related to their behavior. For example, a turtle sprite might have visual semantics in terms of having a head and a tail, such that the sprite should (normally) move in the direction of its head. The sprite's semantics suggest that the sprite ought to rotate as needed so that the sprite is properly oriented before movement commences. Thus, a distinctive problem of animation is to combine rotation and translation operations in a way that yields semantically consistent behavior. Some animations include this rotation; others omit it. During our informal review and in our prior study of a smaller sample of animations (Dahotre *et al.*, 2010), we observed that when primitives are used, they are generally used correctly. We therefore quantify this form of finesse (and those described below) by counting the primitives used for rotation and translation of sprites (the *move* category of Table 1).

Second, we have observed that many animations respond to user input. For example, when a game receives "left-arrow" keystrokes from the user, it might move a sprite to the left. The problem is how to implement the code for collecting inputs and processing events. Scratch makes it possible to do this in a highly modular way by associating an event handler for each kind of input. We can therefore quantify this form of finesse in terms of the number of primitives used for collecting user inputs (the *oninput* category in Table 1).

Third, we observed that animations often had discernable segments. For example, story-telling animations often contained multiple scenes, and games often had distinctive levels. At the opening of each story scene or at the beginning of a game level, new sprites should appear. At the end of the scene or level, sprites should disappear. Thus, another distinctive problem of animation is to invoke primitives for making sprites appear and disappear. However, some animations had little apparent finesse in this area, as they omitted instructions for hiding sprites the end of the animation. Appearance and disappearance can be achieved by show or hide primitives, or by changing the sprite's visual appearance (e.g., shrinking to a zero size or changing the image to match the background). We quantify this form of finesse by counting primitives in the *show*, *hide*, and *changelook* categories (Table 1).

Fourth, we observe that virtually all game animations require testing whether sprites have collided. For example, when a player sprite intersects with a bullet sprite, the game might end. Even non-game animations frequently test for collision in order to coordinate sprite movements. In prior work, we even identified a design pattern by which this is commonly implemented: one sprite tests for a collision, and it then issues event messages to other sprites to notify them so that they can take an action in response. Thus, a key problem is to detect the collision and manage related events, which we quantify by counting the number of collision-detection and event-management primitives used (the *pos-read*, *touching*, and event-oriented categories in Table 1).

3.3 Abstractions: Progression in use of macros, imperative code and data

To complement the user sophistication model of technical programming skills, we also considered a model derived in our own prior research. In this work, we surveyed over 800 information workers about what features they or their subordinates used in spreadsheets, scripts, and other programming environments (Scaffidi *et al.*, 2006). We focused on features related to abstraction creation, due to abstractions' role in raising code quality (Scaffidi *et al.*, 2005). For example, we asked about creation of macros, functions, templates, database tables, stored procedures, and charts. Factor analysis revealed three clusters of features. The first included features for creating macros (typically through programming-by-demonstration). The second included features for textual imperative code (such as functions). The third cluster included features specifically for manipulating data (often in a linked structure such as database tables linked by keys). Users who were inclined to use one feature also were inclined to use other features in the same cluster.

For each cluster, we categorized people based on whether they had a high or low propensity to use features in that cluster. In combination with other information that we collected from respondents, we used our categorization to test for statistically significant differences between people. This analysis revealed numerous interesting differences. For example, compared to people with a low propensity to use imperative coding features, people with a higher propensity also reported a higher tendency to test their programs, a higher tendency to create documentation, and greater programming knowledge. These findings suggested that our model was useful for revealing meaningful differences in skill.

Adapting the model to Scratch

As with finesse (Section 3.2), we can quantify usage of Scratch abstractions by counting primitives without relying on a user survey. Of the three clusters, macros are irrelevant in the Scratch context because PBD is not supported. While Scratch does not support functions, the other features in our imperative cluster were essentially generic coding primitives, corresponding most closely to those in Scratch for event-oriented programming and basic flow of control (Table 1). The third feature cluster is most closely related to Scratch's *datamanip* primitives for reading and writing variables and arrays. (We note in passing that these abstraction primitives closely matched those reported in the study of afterschool programming (Maloney *et al.*, 2008).) Overall, these abstraction-related primitives and those measured to assess finesse (Section 3.2) together cover nearly all of Scratch's primitives (Table 1), so we opted in our study below to count each category of primitives supported by Scratch.

3.4 Scratch history: Progression by coding efficiency and time taken

It is well-known and has been shown in study after study that among professional programmers, the most-skilled people can produce far more lines of code per unit time than the least-skilled, often by an order of magnitude or more (Boehm and Papaccio, 1988; DeMarco and Lister, 1999; Boehm *et al.*, 2000; Oram and Wilson, 2010). Based on these results, models have been developed to predict how long it will take a programmer of a certain skill level to produce software with a given set of functions. Of these models, one of the most widely-validated is COCOMO, which has been in development over 30 years (Boehm *et al.*, 2000). In addition to programmer skill, this model accounts for the effect of programming language, tools, and other factors on the time required to implement a program that must provide a given set of functions. The latest version of COCOMO predicts that a programmer with 3-6 months of experience with a programming language, domain, tool, and related skills will take 17 times as long to implement any functionality as a programmer with 1-2 years of experience (holding all other factors constant). The main reason for this difference is not primarily that better programmers need to write less code for the same amount of functionality; in fact, given a particular programming language, the number of lines of code required to implement a given functional specification is surprisingly constant regardless of the programmer skill (Jones 1995; Boehm *et al.*, 2000). Rather, the primary reason is that better programmers simply need less time to write lines of code. Yet, while it is well-known that a hallmark of more skillful professional programming is the ability to produce code faster, we are not aware of any studies exploring whether users of any animation-programming environment demonstrate a comparable increase in programming speed as they gain experience over months and years. The current study provides an opportunity to explore this issue in the context of Scratch.

Adapting the model to Scratch

To assess how long users took to implement their Scratch animations, we analyzed the animations' save history, which is stored in the animation file and contains data on the date and time of each save as well as the time the file was uploaded to the Scratch website. Using this information, we estimated the total time for each animation, and we estimated efficiency by dividing this total time by the number of primitives used in the animation. Specifically, for each user, we summed the time between save events that occurred on the same day to get a total time on that day. In order to obtain an estimated time for days in which the user only saved once, we computed the average time between save events for that user on the other days where there were multiple save events. We then added these daily estimates to obtain the estimated total amount of time spent on creating each animation. Dividing this by the number of primitives in the animation yielded the estimated time per primitive.

To explore the sensitivity of results to our method of analyzing save history, we tried other methods of using the save history to estimate the time spent coding each animation. For example, in one method, we discarded data from days that had only one save event, thus eliminating the need to rely on average time between save events. In another method aimed at reducing the effect of large intervals between save events (e.g., when a programmer saves an animation at 9am and then again at 7pm) on the inferred time, we experimented with capping the amount of time that was added to the total time for each save event. We are not aware of any way in which any of these methods are biased toward or against users with more or less experience. Consequently, we were not surprised that these alternate methods (and others that we tried) all yielded essentially identical results.

4 Data acquisition

To obtain data for analysis, we wrote a "screen-scraper" program that automatically downloaded animations and their associated project pages from the Scratch website. The program randomly selected animations using the website's "Surprise Me" feature, then added the username of each animation's creator to a set until 250 distinct users had thereby been identified. (Identifying 250 distinct users required looking at 276 different animations.) In order that we could apply the onion model, our scraper also recorded data from the website about when users had created galleries, how many "friends" they had, and which animations had been uploaded to other users' galleries.

In order to perform the detailed primitive-usage analysis required by the other two models, we wrote a program to analyze a sample of animations for each of the 250 Scratch users. For each user, the program paged through the list of the user's Scratch animations on the website (where there are 15 animations per page) and retrieved one animation for each page of animations. For the page with the earliest animations, our program retrieved the oldest animation, so that our program always downloaded the user's very first animation; in addition, the program randomly selected one of the 15 animations on each of the user's other pages. This yielded between 1 and 140 Scratch animations for each of the 250 users, for a total of 2195 animations. For each animation, our program then counted the number of times that each category of primitive was invoked; of the 2195 Scratch animations, 403 could not be parsed in this manner due to version incompatibilities, and we discarded 1 outlier that was extremely large because the Scratch user had simply copied and pasted the same code in the animation several thousand times. This provided primitive-usage statistics for a total of 1791 Scratch animations. Among these, our programs were able to extract save history data from 467 animations (the others of which did not include the requisite meta-data due to tool version, configuration, or because the user never saved any animation more than once, making it impossible to determine the amount of time that should be imputed).

The 1791 Scratch animations were created anywhere from 0 through 34 months after each Scratch user's first published animation. In our analysis, we round time to the nearest month. The median duration of user involvement was 3 months, with a mean of 6.99 months and a mode of 0 months (meaning that the most common case was that a user dropped out at 2 weeks or less). For visual clarity, our graphs below depict the 96% of the Scratch animations that were created anywhere from 0 through 18 months after users' first published animation (Figure 4), since later months contain fewer than 25 animations per month, which resulted in significant visual scatter. Despite this scatter, however, most of the trends discussed are also somewhat present but less obvious beyond the 18th month. All statistical tests below are computed on individual (non-aggregated) Scratch animations from all months, including those beyond the 18th month. In short, our findings should be considered to apply within the first 18 months of Scratch users' experience, with somewhat less confidence in generalizability beyond that period.

5 Results

5.1 Onion model

Using our adapted onion model, we categorized our 250 Scratch users (Figure 3). We found that approximately a third were Passive Users, who created only a handful of animations. Nearly half were Peripheral Users, who created more than a handful of animations but who had not become involved in many galleries. Over a fifth of users were Active Users, meaning that they had contributed to quite a few different galleries. We found only 3 Project Leaders, which was consistent with the prior work which reported that very few people become leaders of open source projects (Ye and Kishida, 2003). We observed few Remixers, which is consistent with our prior work showing low levels of remixing in the Scratch community (Dahotre *et al.*, 2010).

In order to assess whether there were meaningful differences among these categories, we computed statistics over members of each category (Table 2). Across the five categories that we considered, a clear progression became apparent in every statistic. For example, even though the number of Scratch animations was only used to recognize members of the lowest three categories, this statistic continued to rise even for members of the highest two categories. Likewise, even though the number of Scratch galleries created only played a role in recognizing members of the highest category, this statistic rose across all five categories.

Our adapted onion model also revealed interesting trends in another two statistics that we computed but which were not used to categorize Scratch users. First, the *average number of friends* showed a steep upward trend across categories, suggesting a trend of increasing social involvement. Second, members of higher categories have a *higher average duration of activity* than members of lower categories (defined as the time between a Scratch user's first and most recent postings of a project), suggesting that Scratch users started at the lower levels and sequentially progressed over time to higher levels. To verify this inference, we manually performed a historical analysis of what categories Scratch users moved through over the past few years. In order to keep the historical analysis manageable, we focused on the histories of the 34 Active Users who had created no more than 750 animations each. Of these, 16 followed the path Passive User → Remixer → Peripheral User → Active User. The other 18 skipped the Remixer category.

Overall, our analyses confirm that Scratch users indeed follow an onion-model-like progression in their kind and intensity of activities. As a waypoint along this progression, remixing of Scratch animations might play a similar role to open source bug fixing. Moreover, 1% of Scratch users were Project Leaders who created many galleries while 21% were Active Users who contributed to many galleries, which shows a nearly identical level of leadership compared to the 1% and 25% reported for analogous statistics in one healthy open source community (Ye and Kishida, 2003).

5.2 Breadth and depth

We plotted the breadth and depth of animations as a function of the number of months that had elapsed between each Scratch animation's creation and the time when the same user first published an animation on the website. Based on the afterschool study, we expected that online Scratch users would demonstrate an increase in sophistication over time, with rising breadth and depth demonstrated by Scratch animations.

In contrast, we found that the average depth and breadth demonstrated by Scratch users' animations actually *decreased* over time (Figure 5). The progression was not monotonic yet was particularly visible for our measure of breadth, the number of distinct categories of primitives. A linear regression is significantly different from zero ($P < 0.01$) and it indicates a downward slope of 0.05 primitive categories per month. For depth, the downward progression is not so visible and not statistically different from zero (at $P<0.05$).

Puzzled by the fact that early Scratch animations demonstrated more sophistication than later animations, we manually investigated a sample of users' early animations. We identified and explored four possible explanations for the downtrend in sophistication that we observed.

First, we found that 149 of the 1791 animations were revised and republished after their first posting. The Scratch website computes age based on when an animation was *first published* by a user, but it displays the *most recent* version of code for the animation. Thus, older animations might hypothetically have been more likely to accumulate new versions and additional complexity over time, making them *seem* more sophisticated than they really were when first created. To test this possible explanation for the observed downtrends in breadth and depth, we repeated our analysis using only the 1642 Scratch animations that could be parsed and which were never revised. This had no visible or statistical effect on the downtrends.

Second, we found that 532 of these 1642 Scratch animations were remixes. That is, the Scratch website reported that these animations were based on code taken from existing animations, rather than being completely new animations. If people hypothetically had a tendency to remix larger animations rather than smaller animations, and if they tended to do relatively more remixing when they were relatively less skilled, then this could have possibly generated the observed downtrends. To test this hypothesis, we filtered Scratch animations to only include the 1110 that were parseable, had no revisions, and that were not based on remixing. We found that breadth and depth still showed downtrends. In fact, the downtrend in depth became even more apparent (Figure 6), and each downtrend's slope was statistically different from zero ($P<0.01$). We also computed the proportion of animation-creation that was based on remixing, and we found that this proportion was fairly constant over time (Figure 7), contradicting our hypothesis that perhaps remixing plays a larger role early on. Thus, remixing apparently does not explain the observed downtrends.

For the remainder of this paper, we restrict our analysis to these 1110 Scratch animations that were parseable, never remixed, and not based on remixing, since these appear to us to be the cleanest representation of skill progression.

Third, we considered whether the complexity of Scratch animations has been decreasing in the community as a whole over the years. If this hypothetical community-level effect were large enough, it could overwhelm any individual learning and possibly cause the observed downtrends in depth and breadth. To explore this hypothesis, we plotted and regressed breadth and depth against the publication date of animations. We found that over the history of the community, both breadth and depth have been slowly rising (with slopes of 0.0014 / month and 0.032 / month, respectively), though the regression is not statistically significant ($P<0.01$). If the skill demonstrated by the community as a whole is not falling over time, then community effects cannot account for the downtrends that we observed at the level of individual users.

Fourth, we considered whether the downtrend might be attributable to earlier dropout by more skilled users. That is, we hypothesized that perhaps people do demonstrate improved skills during the period of time that they use Scratch, but that perhaps this progression is masked by earlier dropout by users with more programming skill (whose departure from the community would progressively pull averages down over time in Figure 6). To test this hypothesis, we focused on the 145 people who published animations in more than one month; for each person, we computed the average breadth and depth in the first and last month of activity. We then performed a pairwise two-tailed t-test to test whether the means for these months were significantly different. The result was inconclusive (Table 3). Both breadth and depth apparently increased for individuals over the duration of their activity, but this change was not quite statistically significant (at P<0.05). We found similar results when we repeated the test, each time using a particular month (such as 4 or 12) and comparing each active user's animations during that month to the animations created in that user's first month. As a final test of our hypothetical explanation that skilled Scratch users drop out earlier than less skilled Scratch users, we regressed dropout time against the depth of each user's first animation, but the result was not even close to statistically significant (at P<0.05), again failing to confirm our hypothesis that earlier dropout by more skilled people caused the observed downtrends.

In short, users in the online Scratch community demonstrated decreasing levels of breadth and depth as they gained experience with Scratch. These downtrends might be explained partially, but not completely, by an early loss of more skilled users from this community.

5.3 Finesse and abstraction-creation

Using the 1110 Scratch animations that were parseable, had no revisions, and that were not based on remixing, we proceeded to compute whether primitive-usage showed any progression over time. In particular, for each category of primitives supported by Scratch (Table 1), we computed the average number of primitives used in animations from each month. We then performed a linear regression for each category of primitives.

We found that the average primitive usage per Scratch animation went down or stayed the same for every single category of primitive (Table 4). The decrease was statistically significant for 6 of the 13 categories that were identified as particularly interesting on the basis of finesse or abstraction creation.

Although these analyses revealed a decreasing use of Scratch primitives that pervaded virtually every category, there were a few exceptions where categories did not demonstrate much of a downtrend at all. First, the *hide* and *show* primitives had a particularly high P value, indicating an absence of a statistically significant downtrend in this area. As noted earlier, these primitives are particularly useful at the start and conclusion of segments in animations, such as scenes in stories and levels in games. This is accomplished at approximately the same rate in later animations as in earlier animations. Second, the *onmessage* and *sendmessage* primitives also showed no statistically significant downtrend. These primitives play a role in custom event-firing, as when coordinating multiple sprites. This appeared to be as prevalent in later animations as in earlier animations.

5.4 Time spent and user efficiency

Using the 467 Scratch animations that were parseable, had no revisions, that were not based on remixing, and that had useable history data, we computed the estimated total time spent on each animation as well as the ratio of this time to the number of primitives in the Scratch animation. We then performed linear regressions on the data for these two statistics versus user experience.

For the first statistic, we found an upward slope of 21 minutes per month ($P<0.01$), indicating that users spent an increasing amount of time on animations as they gained experience. For the second statistic, we also found an upward slope of 6.57 seconds per programming primitive, though this was not statistically significant (at $P<0.05$), indicating that users became slightly (but not significantly) slower as they gained experience. These results indicate that users did not become faster at writing programs over time—specifically, we certainly did not observe the order-of-magnitude increase in efficiency common among professional programmers (Section 3.4).

In order to understand these results, we considered whether Scratch users simply were increasing in efficiency in the sense of being able to produce more functionality with the same number of primitives. In other words, perhaps more experienced users were spending more time per animation, yet simultaneously using equal or fewer primitives per animation (Section 5.2), because they were able to use primitives more parsimoniously and create more functionality (in more time) without needing more primitives to do so.

To test this hypothesis, we examined all 37 Scratch animations created in the 17th or 18th month of the any user's experience and, for each of the corresponding 16 users, also examined the user's first animation posted to the website. (As above, we focused on animations that were parseable, not revised, and not a remix.) We sought to qualitatively identify any way in which 17th/18th-month animations demonstrated more functionality than these users' first animations.

Yet we could not find any systematic difference at all. In fact, if anything, the 17th/18th-month animations provided *less* functionality than the users' first animations. For example, one 17th/18th-month animation simply showed three yellow dots that silently bounced along the contour of a shape; the corresponding user's first animation had music and an interactive screen that accepted mouse input to select which weapon an animated character should carry. As another example, one 17th/18th-month animation showed two grey shapes, with no animation; the corresponding user's first animation had a fairy with animated wings. Of the 17th/18th-month animations, only 24% had any interactivity, only 43% had any sound, and only 32% had animation. In contrast, of the users' first animations, 69% had interactivity, 50% had sound, and 75% had animation.

In short, we did not find any increases in the efficiency of Scratch users as they gained experience. They did not become faster in terms of amount of time per programming primitive written, and they did not appear to create more functionality in the time that they did spend.

6 Discussion and conclusions

In this paper, we have answered the question, *To what extent do users in the Scratch environment demonstrate increasing skills over time?* Considering that Scratch is so widely-used, well-respected and previously-successful, we anticipated that we would observe a positive progression of one or more skills demonstrated in animations uploaded by users to the Scratch website. What we found was a positive progression of social skills (Section 5.1) alongside a negative progression of demonstrated technical expertise, as reflected in a wide range of programming primitives (Sections 5.2 and 5.3). This downtrend was apparent even after we applied several filters, including omitting animations that were ever revised, as well as omitting animations that were remixes of old code. The downtrend might be partially, but not completely, explained by early loss of more skilled users from the community. There was also a flat or possible downtrend in how efficient Scratch users were (Section 5.4). In addition to the downtrend in usage of programming primitives, we also observed a rather high level of drop-out, with 3 months being the median duration of involvement (Section 4).

One possible interpretation of our results is that users do not, in fact, increase in technical programming skill over time while using Scratch. Perhaps the tool+community approach, at least as applied in the Scratch environment, is not effective at increasing users' ability to use elementary programming primitives, or their ability to work more efficiently.

However, two other possible interpretations are possible, which we discuss in detail below. One is that Scratch users are indeed learning, but that this learning must be measured in ways other than those that we have explored. Another possible interpretation is that Scratch users are learning, and indeed even learning on the measures that we have used, but that more experienced users are less likely than inexperienced users to demonstrate their skills.

Future work aimed at investigating other possible measures of skill

In the current study, we have investigated how four models of demonstrated skill might shed light on skill development within the Scratch environment. Within these models, most of the measures (breadth, depth, finesse, abstractions, efficiency) were strongly tied to usage of Scratch primitives, which was precisely the same piece of evidence used to argue in related work that Scratch users were mastering elementary programming skills through the Scratch clubhouse (Maloney *et al.*, 2008).

Yet other measures of skill might reveal forms of learning that we did not explore. For example, one measure that could be investigated is the rate at which Scratch users create bug-free animation programs; this could be assessed by interviewing users about their own animations or about animations that other people have created. Another measure of skill might be the ability to create reusable code, and this could be assessed by analyzing the number of times that other people remix each Scratch animation. Still other measures might assess the extent to which Scratch animations demonstrate effective use of advanced animation concepts, such as specific temporal behaviors and use of geometric constraints.

Other measures could be developed around the idea of assessing how well users perform at different levels (Rasmussen 1983). At the lowest level, measures could assess "sensory-motor performance which… take place without conscious control as smooth, automated, and highly integrated patterns of behavior" (Rasmussen 1983)—for example, how well are Scratch users able to drag and drop primitives into scripts, and does their performance change with experience? At a higher level, measures could assess what rules are users able to demonstrate that they have mastered, and at a still higher level, measures could assess what goal-setting, planning, and problem-solving abilities the users demonstrate. For example, would Scratch users demonstrate increasing ability to accurately decide when to use loop primitives versus other primitives, as well as the ability to make overall plans for constructing animations?

Future work aimed at investigating reasons why users might not demonstrate their skills

Another possible interpretation of our results is that perhaps Scratch users' technical programming skill is constant or increasing on average, but that they demonstrate this skill progressively less over time. The question would then become, "Why are Scratch users disengaging and what could be done to change this?" There are several possible causes for disengagement, including intimidation by the Scratch environment, distraction by other community members, and boredom with the tool's capabilities.

The first possible cause of disengagement, intimidation, is a reasonable hypothesis considering how the online Scratch repository is structured: the very first thing that users see through the homepage is a list of the most highly-rated, complex, full-featured animations on the website. In contrast, studies have shown that people actually learn faster if they are provided with relatively simple training materials first and then gradually progress to more complex materials (Jackson *et al.*, 1998; Hamade *et al.*, 2005; Sharma and Hannafin, 2007). One study, in particular, found that approximately two-thirds of learners in an online environment required significant guidance in selecting and navigating to training materials that were relevant in terms of topic and level of difficulty (Chen and Macredie, 2004). If Scratch users are dropping out due to intimidation by the complexity of example animations provided to them, then perhaps dropout could be alleviated by restructuring the website to give users a more graduated introduction that progresses in complexity as users gain experience over months or even years.

Another possible cause of disengagement, distraction, is interesting because our prior studies have found that a sizeable amount of online conversation between Scratch users is, in fact, about unrelated topics (e.g., video games) (Dahotre *et al.*, 2010). This explanation would also be consistent with the fact that our current study revealed that Scratch community members tend to reach higher levels of social interaction over time. If this explanation turns out to be true, then the community part of the Scratch tool+community approach could in fact be somewhat detrimental rather than fully supportive for the development of technical skills. Future research could further test this hypothesis in the context of Scratch or other environments.

A third possible cause of disengagement, boredom, might result from the limited range of different programs that can be created with the Scratch environment. Insights for how to reduce boredom might be motivated by established theories such as Innovation Diffusion Theory, which describes the factors that are commonly necessary before people choose to permanently assimilate a technology (such as Scratch) into their lives. For example, a technology is more easily assimilated if it is compatible with existing complementary technologies and if it has an easily verifiable advantage over existing competitive technologies (Rogers, 1995). In terms of compatibility, Scratch is currently not interoperable with any other widely-used technologies (e.g., email, spreadsheets, iPods), except for the fact that animations can be embedded as applets in web pages. In terms of relative advantage, it might not be obvious to users what advantages Scratch has over other technologies that could be used to create animations (e.g., PowerPoint or Flash) or that could be used to creatively exercise programming skills for new purposes (such as using Scratch to show presentations instead of PowerPoint, or using Scratch to enhance websites instead of Flash). Enhancing Scratch along these and other dimensions might substantially increase the number of people who continue using it beyond a brief trial period.

One final approach for reducing boredom might leverage the community in order to increase the intensity and frequency of Scratch users' online activities. At present, the Scratch environment provides few incentives to drive users toward creating increasingly sophisticated animations. This stands in contrast to the afterschool club setting, where the Scratch designers report, "Every two or three months, we organized a Scratch-a-thon during which all clubhouse members would work on Scratch for 3-4 hours and share publicly the projects they had created" (Maloney *et al.*, 2008). As a result, students in the club might have felt a sense of urgency and a desire not to be embarrassed in front of their friends, thereby driving them to produce increasingly impressive animations; in contrast, online Scratch users can take as long as they want to post an animation, and they might not feel much shame in failing to impress their anonymous peers. Outside of animation programming, other researchers have found that particular activities, such as programming contests, can be useful for motivating people to write and share Matlab code (Gulley, 2004). Future work could evaluate whether similar contests and other incentive schemes would help keep users engaged and actively learning in the Scratch environment.

7 References cited

Boehm, B., and Papaccio, P. (1988) Understanding and controlling software costs. *IEEE Transactions on Software Engineering, 14*(10), 1462-1477.

Boehm, B., Abts, C., Brown, A. W., Chulani, S., Clark, B., Horowitz, E., Madachy, R., Reifer, D., and Steece, B. (2000) *Software Cost Estimation with COCOMO II,* Prentice Hall.

Chen, S. Y., and Macredie, R. D. (2004). Cognitive modeling of student learning in web-based instructional programs. *International Journal of Human-Computer Interaction, 17*(3), 375-402.

Cooper, S., Dann, W., and Pausch, R. (2000). Developing algorithmic thinking with Alice. *Information Systems Educators Conference*, 506-539.

Cypher, A., Smith, D, and Tessler, L. (2001). Novice programming comes of age. *Your Wish is My Command*, Morgan Kaufmann, 7-20.

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*, 1-10.

DeMarco, T., and Lister, T. (1999) *Peopleware: Productive Projects and Teams,* Dorset House.

Gulley, N. (2004). In praise of tweaking: A wiki-like programming contest. *Interactions*, *11*(3), 18-23.

Hamade, R., Artail, H., and Jaber, M. (2005). Learning theory as applied to mechanical CAD training of novices. *International Journal of Human-Computer Interaction*, *19*(3), 305-322.

Hofmann, B, and Wulf, V. (2003) Building communities among software engineers: The VISEK approach to intra-and inter-organizational learning. *Advances in Learning Software Organizations*, 25-33.

Huff, S., Munro, M., and Marcolin, B. (1992). Modelling and measuring end user sophistication. *Proceedings of the 1992 ACM SIGCPR Conference on Computer Personnel Research*, 1-10.

Jackson, S, Krajcik, J, and Soloway, E. (1998) The design of guided learner-adaptable scaffolding in interactive learning environments. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 187-194.

Jones, C. (1995) Backfiring: Converting lines of code to function points. *Computer, 28*(11), 87-88.

Koh, K., Basawapatna, A., Bennett, V., and Repenning, A. (2010). Towards the automatic recognition of computational thinking for adaptive visual language learning. *Proceedings of the Visual Languages and Human-Centric Computing*, 59-66.

Maloney, J., Peppler, K., Kafai, Y., Resnick, M., and Rusk, N. (2008). Programming by choice: Urban youth learning programming with scratch. *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, 367-371.

Mebane, M, Porcelli, R, Iannone, A, Attanasio, C, and Francescato, D. (2008) Evaluation of the efficacy of affective education online training in promoting academic and professional learning and social capital. *International Journal of Human-Computer Interaction, 24*(1), 68-86.

Monroy-Hernández, A., and Resnick, M. (2008). Empowering kids to create and share programmable media. *Interactions, 15*(2), 50-53.

Moskal, B., Lurie, D., and Cooper, S. (2004). Evaluating the effectiveness of a new instructional approach. *ACM SIGCSE Bulletin, 36*(1), 75-79.

Munro, M., Huff, S., Marcolin, B., and Compeau, D. (1997). Understanding and measuring user competence. *Information & Management, 33*(1), 45-57.

Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K. and Ye, Y. (2002). Evolution patterns of open–source software systems and communities. *Proceedings of International Workshop on Principles of Software Evolution.*

Oram, A., and Wilson, G. (2010). *Making Software: What Really Works, and Why We Believe It,* O'Reilly Media.

Pane, J. (2002). *A Programming System for Children that is Designed for Usability*. PhD Dissertation, School of Computer Science, Carnegie Mellon University.

Papert, S. (1980). *Mindstorms*, Basic Books New York.

Rasmussen, J. (1983) Skills, rules, and knowledge; signals, signs, and symbols, and other distinctions in human performance models. *IEEE Transactions on Systems, Man, and Cybernetics, 13*(3), 257-266.

Repenning, A. (1993). *Agentsheets: A tool for building domain-oriented dynamic, visual environments*. PhD Dissertation, Dept. of Computer Science, Univ. Colorado-Boulder.

Resnick, M. (2007). Sowing the seeds for a more creative society. *Learning and Leading with Technology, 35*(4), 18.

Resnick, M., et al. (2009). Scratch: Programming for all. *Communications of the ACM, 52*(11), 60-67.

Rogers, E. (1995). *Diffusion of Innovations*, The Free Press, New York.

Scaffidi, C., Ko, A., Myers, B., and Shaw, M. (2006). Dimensions characterizing programming feature usage by information workers. In *Proceedings of the Visual Languages and Human-Centric Computing*, 59-64.

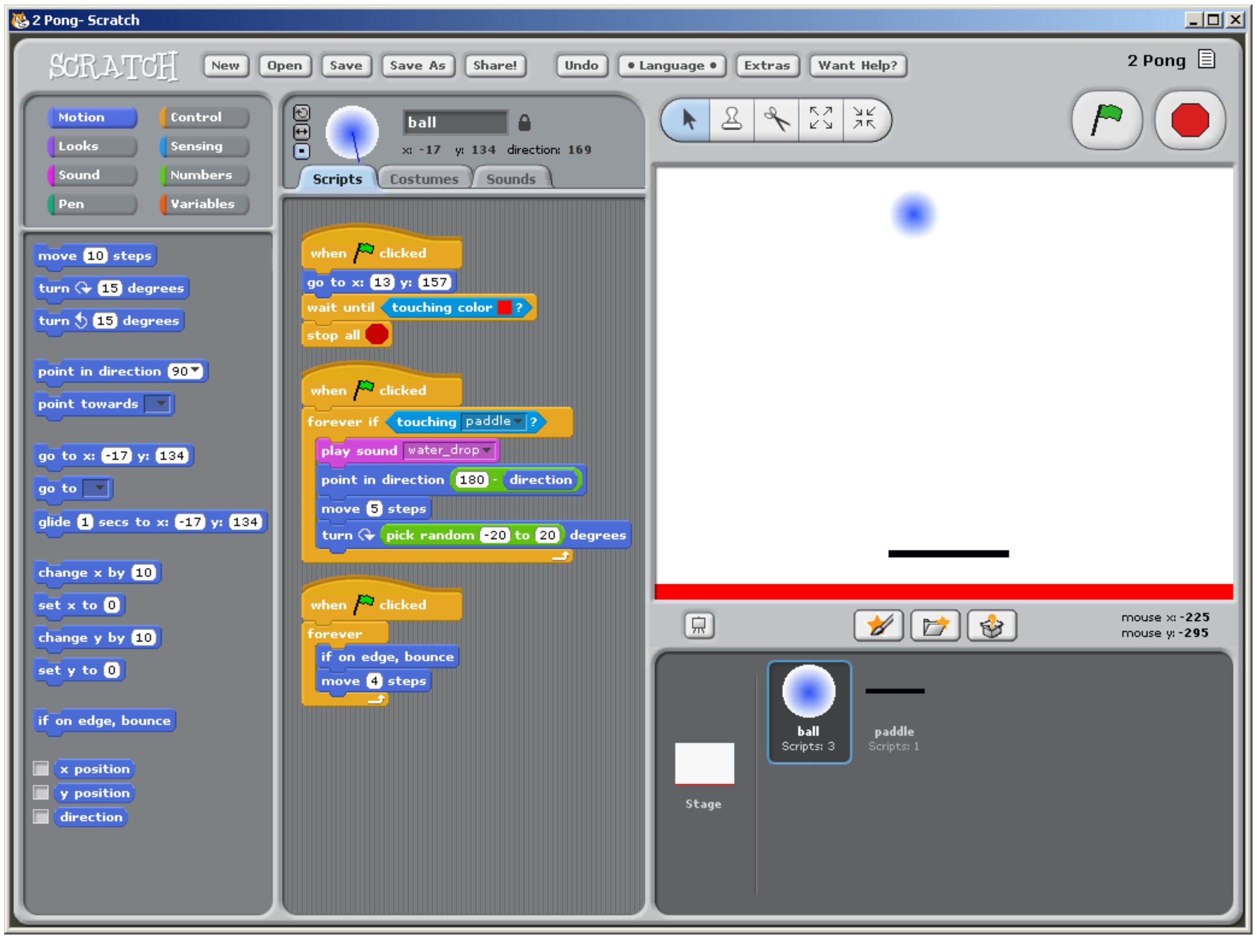
Scaffidi, C., Shaw, M., and Myers, B. (2005). An approach for categorizing end user programmers to guide software engineering research. In *Proceedings of the First Workshop on End-User Software Engineering*, 1-5.

Sharma, P, and Hannafin, M. (2007) Scaffolding in technology-enhanced learning environments. *Interactive Learning Environments, 15*(1), 27-46.

Stahl, G., Sumner, T., and Repenning, A. (1995). Internet repositories for collaborative learning: Supporting both students and teachers. In *The First International Conference on Computer Support for Collaborative Learning*, 321-328.

Wang, M, Poole, M, Harris, B, and Wangemann, P. (2001) Promoting online collaborative learning experiences for teenagers. *Educational Media International, 38*(4), 203-215.

Ye, Y., and Kishida, K. (2003). Toward an understanding of the motivation open source software developers. In *Proceedings of the 25th International Conference on Software Engineering*, 419-429.



```
when  clicked
go to x: 13 y: 157
wait until touching color 🟥 ?
stop all

when  clicked
forever if touching paddle ?
    play sound water_drop
    point in direction 180 - direction
    move 5 steps
    turn pick random -20 to 20 degrees

when  clicked
forever
    if on edge, bounce
    move 4 steps
```

Figure 1. Editing the three scripts for the ball in a pong game. The programmer lays out sprites on the right; clicking a sprite brings up its scripts for editing in the center. Primitives can be dragged-and-dropped from the toolbox at left

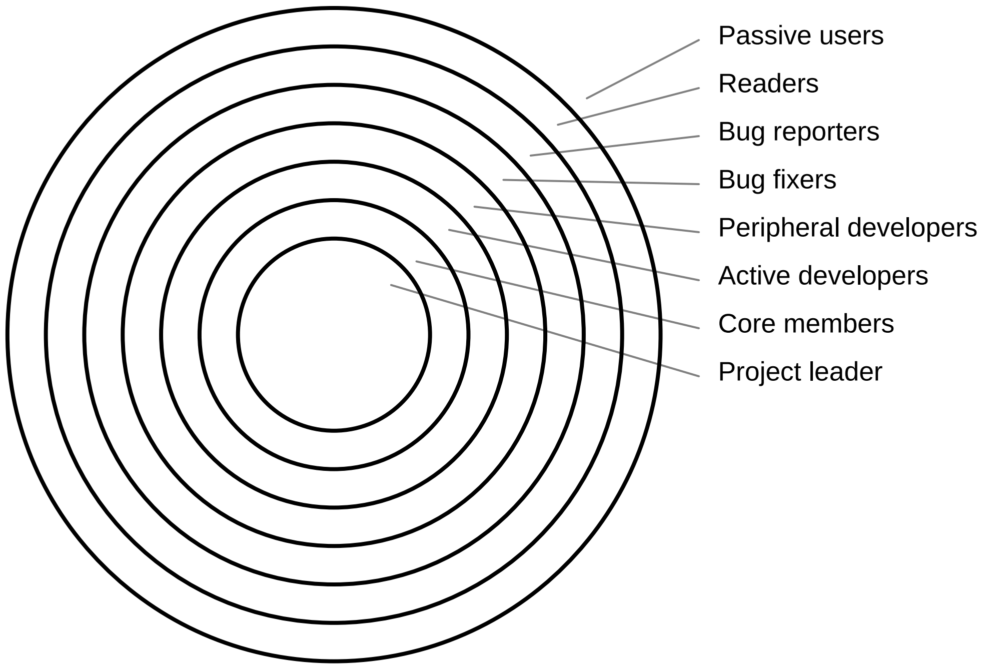


Figure 2. Classification of people in the onion model

Table 1. The 17 categories of elementary programming primitives supported by Scratch.

Category	Description of primitives in this category	Examples
Three categories related to changing appearances of Sprites		
changelook	Changes graphical image of a sprite	Change “costume” (the sprite’s image); change background; resize
hide	Make sprite invisible	Hide; go back behind layers
show	Make sprite visible	Show; go forward on top of layers
Three categories related to motion and coordination of sprites		
move	Translates or rotates sprite	Change direction; move forward; bounce off edge of animation area
pos-read	Read position of this or other sprites	Retrieve sprite’s x or y position; retrieve distance to another sprite;
touching	Detect sprite intersection	Detect if two sprites collide; detect if a sprite is touching any pixels of a certain color
Two categories related to data retrieval and mathematical computation		
datamanip	Manipulating variables and arrays	Read/write variable; read/write array
math	Compute mathematical operation	Add; multiply; or; and; not; random; round; absolute value
Three categories related to event-oriented programming		
oninput	Handle user input event	On keystroke, run specified script; on mouse click, run specified script
onmessage	Handle custom event	On event, run specified script
sendmessage	Fire custom event	Broadcast event; broadcast and wait
Three categories related to basic flow of control		
conditional	Perform statements depending on whether a specified condition holds	If/then; if/then/else
loop	Iterate	Do forever; do until condition
wait	Pause execution for a time	Pause for N seconds
Three categories related to other specialized Scratch features		
sound	Control audio output	Play recorded sound; play midi sound; set volume; set tempo
speak	Display words in bubble over sprite	Show words in speech bubble; show words in thought bubble
other	Everything else	Draw with pen; terminate animation; read inputs from hardware sensors

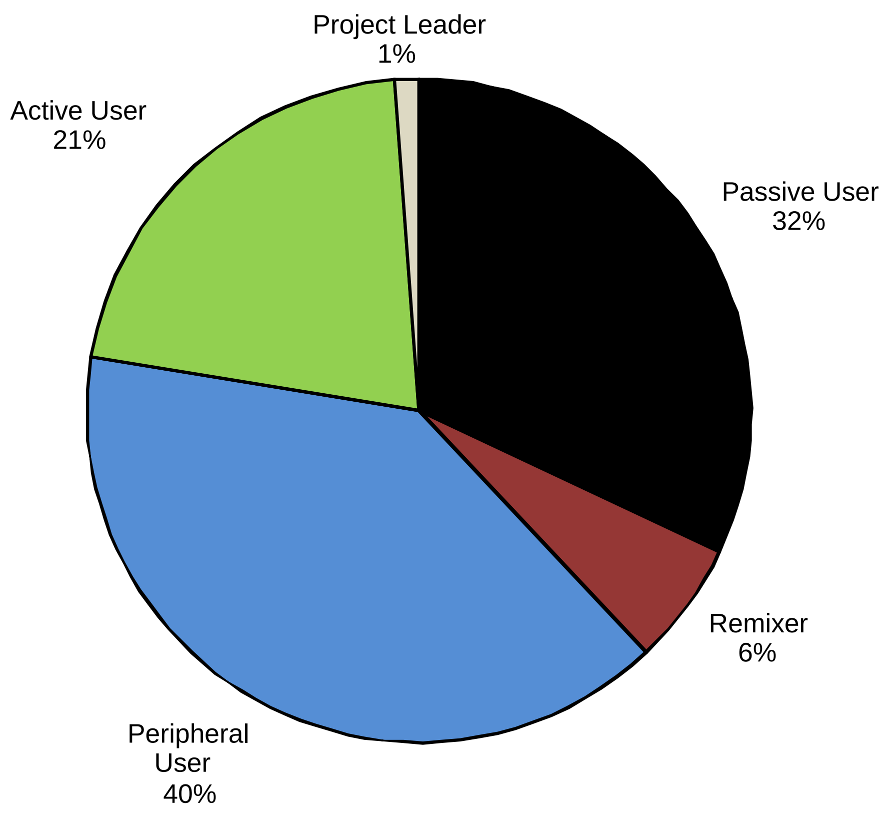


Figure 3. Categorization of Scratch users using our adapted onion model

Table 2. Statistics revealing progression among Scratch users in our adapted onion model.

Statistic (averages)	Passive Users (80 people)	Remixers (15 people)	Peripheral Users (99 people)	Active Users (53 people)	Project Leaders (3 people)	Everybody (250 people)
# animations created	2.7	25.1	64.3	293.6	905.0	100.9
# galleries created	0.1	0.7	2.2	5.7	55.0	2.8
# galleries containing their projects	0.4	1.3	5.3	43.2	44.7	11.9
time span of animation creation dates (months)	1.6	4.3	10.5	15.4	18.3	8.4
# friends	1.6	4.2	21.2	139.7	936.3	50.0

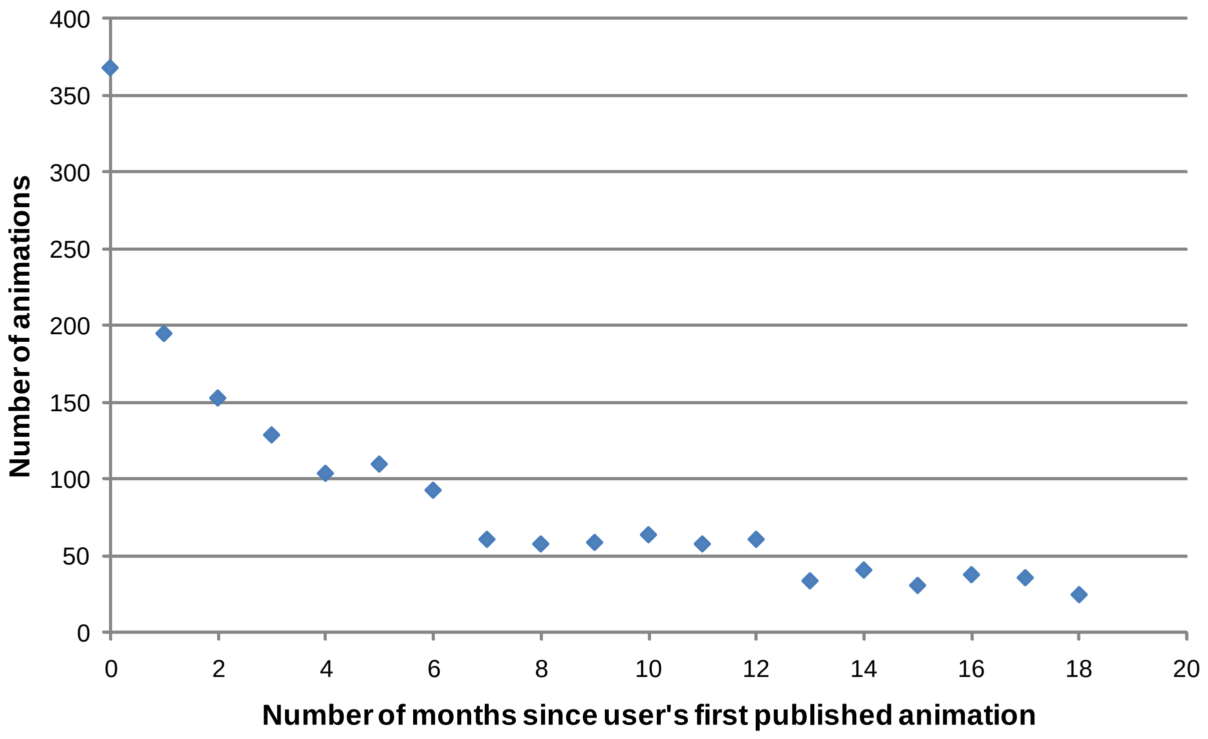


Figure 4. Distribution of Scratch animations analyzed for sophistication and abstractions

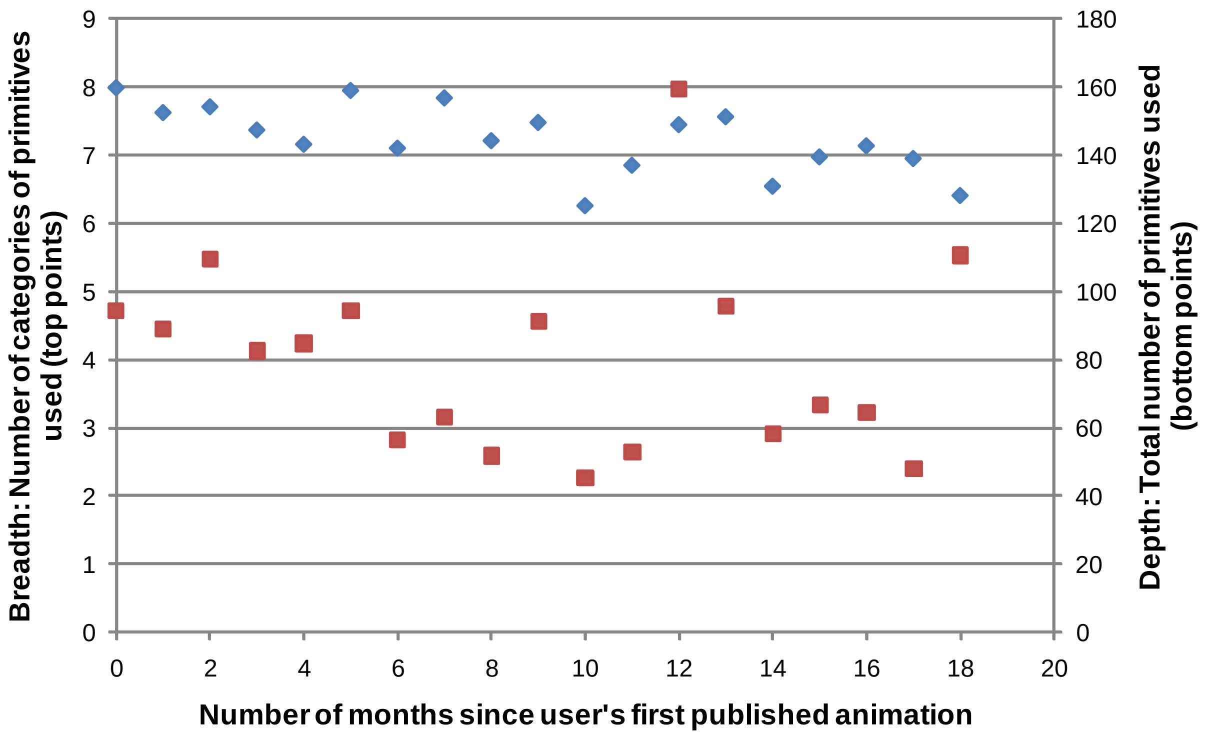


Figure 5. Average breadth and depth of Scratch animations, as a function of time since each animation's user had first published an animation.

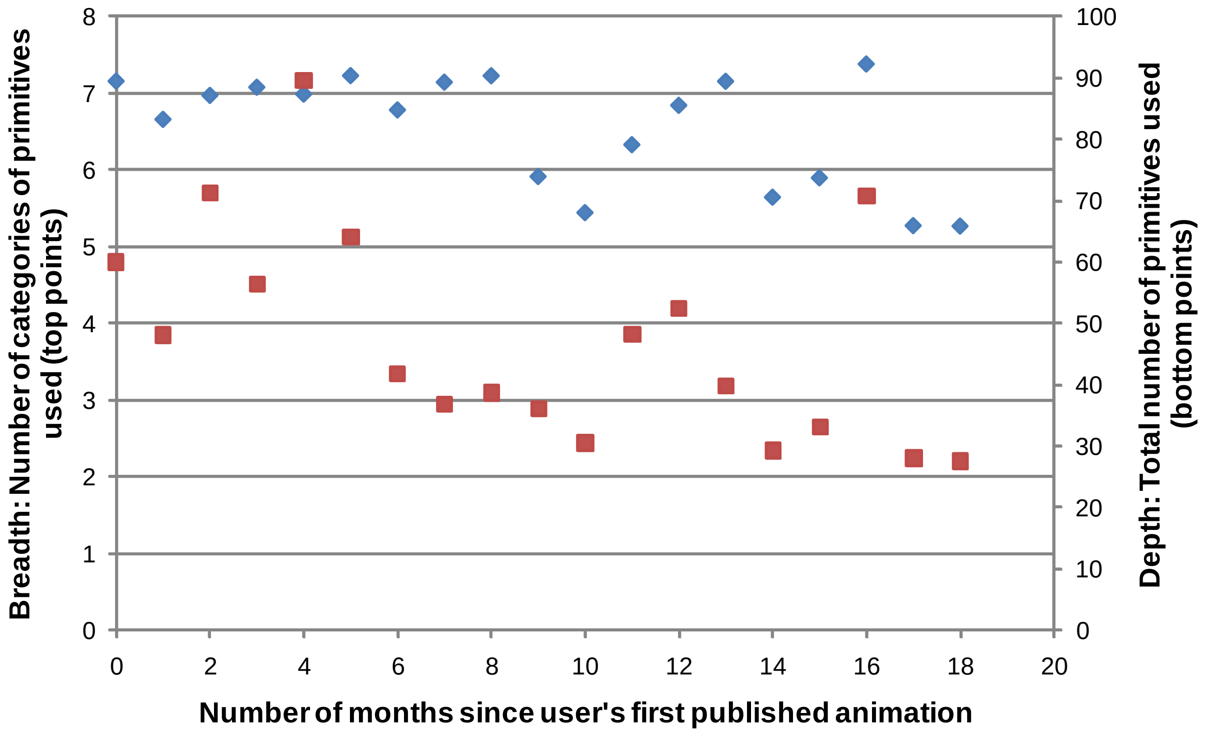


Figure 6. Average breadth and depth of Scratch animations (including only those that were parseable, not revised and not created through remixing), as a function of time since each animation's user had first published an animation.

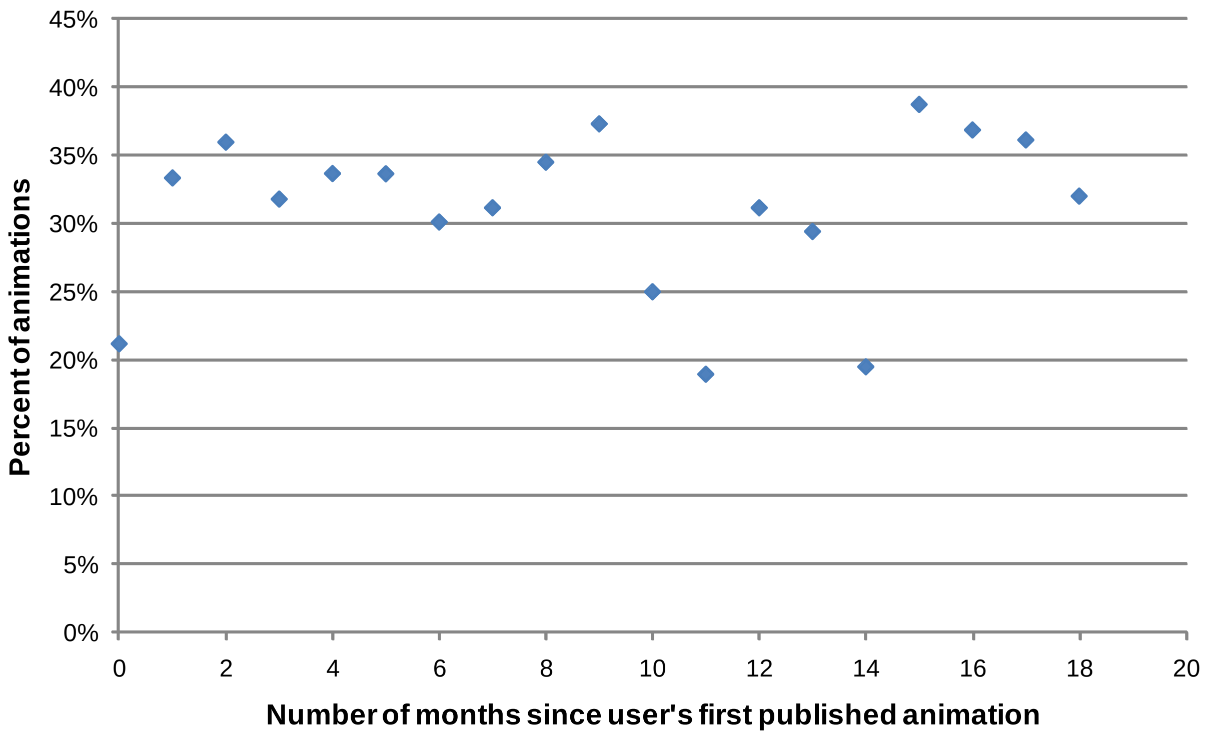


Figure 7. Proportion of all Scratch animations that were created through remixing, as a function of time since each animation's user first published an animation.

Table 3. Pairwise two-tailed t-tests of whether depth and breadth changed for the 145 Scratch users who published animations in more than one month.

	First month	Last month	t	P
Depth	70.40	145.50	1.66	0.05
Breadth	7.74	8.07	0.87	0.19

Table 4. Regression of primitive-usage counts, per Scratch animation, as a function of months since each user's first published animation. Significant tests at P<0.05 are bolded (indicating a two-tailed test of whether the slope is nonzero), along with their corresponding slopes.

Category	Animation finesse	Abstraction model	Slope	Slope t value	Slope P value
changelook	x		-0.15	-1.88	0.06
hide	x		-0.01	-0.20	0.84
show	x		-0.03	-1.16	0.25
move	x		**-0.20**	-3.35	**<0.01**
pos-read	x		**-0.07**	-2.62	**<0.01**
touching	x		**-0.08**	-3.61	**<0.01**
datamanip		x	-0.15	-1.80	0.07
math			**-0.18**	-2.20	**0.03**
oninput	x	x	**-0.13**	-2.96	**<0.01**
onmessage	x	x	-0.02	-0.21	0.84
sendmessage	x	x	0.00	-0.09	0.93
conditional		x	**-0.12**	-3.06	**<0.01**
loop		x	**-0.11**	-2.42	**0.02**
wait		x	-0.10	-1.52	0.13
sound			-0.04	-1.86	0.06
speak			**-0.07**	-2.14	**0.03**
other			**-0.05**	-2.07	**0.04**

Total primitives used (depth)	**-1.51**	-2.76	**<0.01**
Distinct categories used (breadth)	**-0.07**	-3.84	**<0.01**