An Exploratory Study of the Usage of Different Educational Resources in an Independent Context

Wint Hnin, Michelle Ichinco, Caitlin Kelleher
Department of Computer Science and Engineering
Washington University in St. Louis
St. Louis, Missouri, USA
Email: {hnin, michelle.ichinco, ckelleher}@wustl.edu

***Abstract*—There are a variety of learning resources with the potential to support children in learning programming independently. While many of them have been evaluated in laboratory settings, we know little about how children choose to use these resources on their own. We conducted a study organized around a film festival to explore children's open-ended use of four different learning supports: tutorials, code puzzles, in-application documentation and code suggestions. The study began with a workshop to introduce the programming environment and available tools, continued through two weeks of home use, and culminated in a film festival. Results suggest that participants leveraged in-context forms of help most frequently, but valued documentation for question-answering and suggestions for opportunistic learning.**

***Index Terms*—novice programming; programming support; tutorials; code puzzles; examples; documentation**

I. Introduction

Learning new skills independently is a fundamental part of software development today, across all skill levels [1]–[3]. Researchers have created and refined a variety of different types of resources designed to support learning. While many of these resources have been evaluated and shown promise in a laboratory setting, we know little about how learners interact with them in the wild. Although learning support is potentially valuable for all programmers, we have chosen to focus on young novice programmers for two reasons, 1) programming is rapidly becoming a form of basic literacy and 2) opportunities to learn programming are still limited within K-12 schools, particularly in the United States.

We conducted a study organized around a film festival in which we asked children between 10 and 15 to create submissions using the Looking Glass programming environment. While working on their projects, participants had access to four different types of learning resources: tutorials, code puzzles, in-application documentation, and static code analysis based suggestions. Based on a combination of log file analysis and survey results we explore the following research questions:

1) How did participants use each type of learning resource?

2) Why did participants make the learning decisions that they did?

3) How did the use of learning resources contribute to program complexity in their completed submissions?

4) Were there differences in usage patterns based on demographics?

Our study provides support for in-context help that assisted participants in working directly on personal goals, with the strongest support for code suggestions. Code suggestions were accessed and used more than other forms of help in raw counts, by percentage of users, and by time spent. However, participants spoke about valuing both documentation and suggestions in different situations. Participants found documentation most helpful when they had a specific question they were trying to answer, while the suggestions were most helpful in supporting opportunistic learning, pointing out programming opportunities that participants did not realize existed. Our results suggest a need to focus on lightweight and in-context forms of help to better support young novices in learning to program.

II. Related Work

Our study is built upon prior work on learning resources and contributes to the area of help seeking strategies and behaviors.

A. Learning Resources

Previous research has developed a variety of learning resources to provide better support for novices in learning topics like APIs, programming concepts, and complex software. We based our implementations of our learning resources on prior work in Documentation, Examples, Tutorials and Puzzles.

1) Documentation: Programmers often struggle when attempting to learn from API documentation, partly due to difficulties locating the documentation for the right API [4]–[6], and inadequate examples [5], [6]. Recent research has tried to improve the usability of API documentation by addressing these problems. To support users in finding appropriate documentation for their needs, researchers have explored augmenting documentation with usage frequency information [7] and by recommending methods to explore based on source code analysis [4], [8]. Others have augmented documentation with more examples by integrating information from question and answer sites [9], [10], automatically linking code examples and documentation [11], and making examples executable [12].

2) Examples: Based on the popularity of example use for learning programming, researchers have designed systems to support example use for both experienced and inexperienced programmers. Systems for experienced programmers extract relevant example code automatically and enable example search within programing environments [13]–[15]. Reuse

support in novice programming environments similarly aids novices in exploring, integrating, and learning from code examples [16], [17]. However, novices may not know what kinds of examples to seek out, so some systems for less experienced programmers include curated sets of examples [18], [19] or suggest examples designed to help novices overcome barriers or gain skills [20], [21]. Finally, studies have explored how to best present examples especially for novices, such as: including multiple similar examples [22], providing pairs of contrasting examples [21], adding subgoals labels to examples [23], and explaining and visualizing code [24], [25].

3) Tutorials: Much of the research on tutorials has focused on helping users to correctly complete a sequence of instructions. Currently, most tutorials fall into one of two categories: 1) static instructions containing images with explanatory text and 2) videos that demonstrate the completion of a specific task [26]. In both types of media, prior research has found that users often struggle to correctly complete a sequence of steps, making errors or skipping steps [27], [28]. Some tutorials systems attempt to prevent errors by providing step by step instructions integrated into the context of an application [29], [30]. Within the space of video tutorials, much of the research has focused on breaking video content into more manageable chunks by automatically pausing to stay in sync with users [31], focusing on short video tips that demonstrate specific features [32], and breaking larger tutorials into a series of single step videos [26], sometimes with multiple performances available [33]. Finally, Ambient Help searches for relevant tutorial content of either type and presents it on a second monitor as users work on self-directed tasks [34].

4) Puzzles: Programming completion puzzles, more commonly called Parsons problems, are one of the early examples of code puzzles, consisting of a set of randomly ordered statements to assemble [35]. Learners assemble the code and get feedback about correctness. Some research builds on this early work by combining Parsons problems with program visualization [36] and exploring techniques for dynamically adapting the difficulty of problems to help struggling students [37]. While not explicitly linked to Parson's Problems, the interest in using code puzzles to help novices learn programming skills has grown dramatically in recent years. Large scale efforts, such as Hour of Code [38], rely on puzzles to teach. Relatively little work has explored how to maximize learning from puzzles. Two studies have explored the design space of puzzles and found initially positive learning results [39]. Another found that incorporating incorrect statements as distractors hurt learning efficiency but not performance [40].

B. Help Seeking Strategies

Our final area of related research concerns how learners use different types of resources when working on their own projects. This is difficult to study directly, however some research has gathered information about programmers' practices through surveys [1], interviews [3] or through queries submitted to a programming related portal [2]. These studies are complemented by a set of lab and classroom studies in which participants were given larger scale tasks and researchers observed their help seeking behaviors [2], [41]–[43]. Taken as a whole, these studies found extensive use of search, largely in response to problems that had arisen. Participants consulted a variety of online resources including FAQs, technical blogs, and code examples. However, it is important to note that for these studies, participants did not have access to diverse kinds of help resources. Consequently, little is known about what kinds of decisions learners might make on their own, given differing types of available help.

III. Methods

Our study was organized around a film festival in order to explore participants' use of learning resources in pursuit of a goal. In sum, the study included an introductory workshop to introduce the programming environment, participants' independent work over the course of two weeks, and the film festival. This study explored what kinds of help learners leveraged independently while working towards an animation for the film festival.

A. System Overview

For this study, participants used Looking Glass [44], a blocks-based programming environment. To construct programs in Looking Glass, learners drag and drop graphical tiles representing methods and select parameter values using menus. The resulting programs control the motions of characters in a 3D virtual world. We used Looking Glass because it contains four distinct types of learning resources designed based on best practices in: 1) tutorials, 2) code puzzles, 3) in-application API documentation, and 4) code suggestions.

1) Tutorials: Traditionally, many novice programming systems have included tutorials to support learning [45]–[47]. The tutorials in Looking Glass, as shown in Figure 1-F, were used in previous research [39], [48] and are inspired by the mixed media approach of mixT [26] as well as similar ones built for Scratch [45]. To access tutorials, users must exit an animation they are working on and choose a tutorial for a specific programming construct and animation. Tutorials contain a sequence of steps necessary to create a particular program. For each step, learners see a textual description of the action to perform and a looping video demonstrating how to perform it. After completing a step, learners progress through the tutorial by pressing "next", as shown in Figure 1-G.

2) Puzzles: Many recent systems for supporting children in learning programming include programming puzzles [38], [39], [49]. The puzzles in Looking Glass have been carefully designed in prior work to enable learning, and are shown in Figure 1-H [39]. Like tutorials, users must close their current project to access puzzles. Users can choose to create a variety of short animations through a puzzle based on the animation or the constructs that the puzzles teach. Once a user selects a puzzle, they drag and drop the code blocks from the bin in order to solve it (see Figure 1-I,J). They can check their work by playing their animation and comparing it against the correctly assembled program via "Play Correct". As the puzzle

Fig. 1. The Looking Glass programming environment (A). We gave participants three activities: 1) program their own 3D animation (*New World* or *My Worlds*), 2) work on a *Puzzle*, or 3) follow a *Tutorial*. When authoring their own animation, participants were presented with *suggestions* (B) to improve their code. If the user clicks on a suggestion (B), they are shown an example (C) in a separate window on how to improve their code. Participants may also utilize the built-in *API documentation* by clicking the '?' button (D) next to each API method. When clicked, a window with documentation for each parameter in the method (E) pops up (only *duration* is shown in the figure). Alternatively, participants may opt to use a *tutorial* (F) by following the step-by-step instructions (G) to recreate an animation within the code editor (A). Lastly, we also gave participants the option to solve *code puzzles* (H). In the puzzles, participants are given a blank program where they reassemble it by dragging unused statements (I) into the puzzle's solution (J).

executes, Looking Glass provides feedback on correctness of the current actions playing to help users resolve errors.

3) Documentation: The documentation we partially based on the supports provided through online API documentation such as Javadoc. However, for programming languages like Java, IDEs provide code completion as a resource for programmers in addition to documentation.The documentation support within Looking Glass is designed to provide similarly easy-access to information about how to use the methods and constructs within the system [50]. To access information for a method, learners can press a '?' button next to each graphical tile as in Figure 1-D. This brings up a new window which shows a series of code examples involving that tile (Figure 1-E). The examples cover all of the available parameters for each method. Learners can play each example independently. The documentation for each method contains previous and next buttons that takes them to the entry for the previous and next method in the API respectively.

4) Suggestions: Finally, a few systems have explored providing programmers with code suggestions designed to improve their programs [20], [21], [51]. The suggestions system within Looking Glass offers tips to users based on the code in their programs, as shown in Figure 1-B,C [21]. When a user executes their program, the system runs static code analysis to find code that could benefit from an unused API or code construct. For example, a user whose program has a character walking a long distance might benefit from learning about the *walkpace* argument that makes characters walk faster. To prevent overload, the system makes at most one suggestion each time the user executes their program.

Users can access suggestions from a panel where they are listed, as shown in Figure 1-B, or light bulb icons that appear on the code blocks. In either case, clicking on a suggestion opens a more detailed view containing contrasting examples, such as slow and fast *walkpaces* (Figure 1-C). Learners can view the code for each example and also watch the animation execute. Finally, users can click a 'Show Me' button that points them to the location of the relevant code block in the interface.

In this study, we used an initial set of suggestions and rules created for an earlier study [50]. They were designed based

on API methods experts used more often than novices. In order to align better with the other resources, we supplemented the set of available suggestions in two ways: 1) we added suggestions for all API methods that were used at least 10% more frequently by experts than novices, and 2) we provided suggestions for constructs, such as the *repeat* loop.

B. Participants

We recruited participants between the ages of 10-15 from a local STEM Mailing List. 84 participants attended the introductory workshop. Of those, 39 participants completed an animation for the film festival. Three others did not complete a film festival submission, but returned their Looking Glass usage data. Two of these participants did not work with Looking Glass after the introductory workshop, so we have excluded those participants. In sum, we analyzed Looking Glass usage data for 40 participants (19 female, 21 male) who had an average age of 11.73 ($SD = 1.3$), and 62.5% of them had more than 3 hours of previous programming experience. There was no limitation on the experience with programming since we are interested in the effect of different programming experience on the use of learning resources. We compensated the participants with $10 Amazon gift cards.

C. Study Procedures

Our study focused on how participants used learning resources in an independent context. The study included an introductory workshop, participants' time at home working with Looking Glass, and a final in-person film festival. As we wanted to explore participants' behavior around learning resources in an open-ended context, all participants had access to all learning resources.

1) Workshop: The introductory workshop serves two goals: 1) introduce the mechanics of creating programs in Looking Glass, and 2) ensure that participants know how to access each learning resource. The workshop took an hour and a half, divided into 40 minutes focused on creating programs in Looking Glass and the rest devoted to the learning resources.

Before getting started, we asked participants to complete a computing history survey that gathered basic demographic information as well as their previous programming experience.

The introduction to Looking Glass included both how to create a scene and how to animate it. We began with a short demonstration showing how to add characters and objects to the 3D scene as well as how to position them within the scene. Participants then had 15 minutes to create a scene for use during the remainder of the workshop. Next, we demonstrated how to write simple, sequential programs by selecting characters and dragging actions into the code editor. Participants then spent the next 15 minutes creating a simple program. Many participants went beyond these basic skills when building their own programs. We collected the initial programs that participants created to serve as an additional measure of prior programming experience.

During the second part of the introductory workshop, we focused on introducing participants to the available learning resources. To streamline the process, we had participants continue to work using the scene and program they had created in the first part of the workshop. We began by introducing the documentation and suggestions, which can be accessed while working on a program. We showed participants where to find both and directed them to complete a simple task using the information found in each. Finally, we asked participants to complete one tutorial and one puzzle focusing on the parallel construct, *Do Together*, which is often the first programming construct that Looking Glass users begin to employ.

Throughout all of the sessions, researchers circulated to ensure that participants were following the directions and staying on the task. Participants also had a reference sheet showing them how to complete each of the steps covered during the workshop. We encouraged participants to take the sheet home with them to serve as an ongoing reference. At the close of the session, we gave each participant a USB key that allowed them to run Looking Glass on their home computer without installing anything. This version of Looking Glass was augmented to store detailed logs of participants' interactions with Looking Glass and present surveys about their behaviors.

2) Film Festival Preparation: Participants had two weeks between the introductory workshop and the film festival to work on their submissions. As they worked, Looking Glass logged information about their use of the system, including which learning resources they were offered and/or interacted with. Additionally, participants completed surveys about their choices surrounding the use of learning resources.

3) Film Festival: The primary goal of the film festival was to serve as a motivation to complete an interesting animation within the two weeks following the introductory workshop. When participants returned for the film festival, we collected their USB keys containing their survey and usage data. We did not collect any additional information during the film festival. We simply showed all of the programs that participants submitted to the assembled audience.

D. Data Collection and Analysis

We logged the actions that users took within Looking Glass and collected several surveys over the course of the study .

1) Access and Usage of Educational Support: We instrumented Looking Glass to log participants' actions within Looking Glass, including their interactions with the four types of learning resources. We used this raw data to compare how participants used the different learning resources. It is important to note that the four resources that participants had available were quite different in nature and designed for different purposes and so making comparisons between them is not straightforward. The documentation and suggestions were both accessible while participants worked on their film festival submissions, where tutorials and puzzles required that they close their current program to open one related to the target learning content. Suggestions are offered to participants when relevant; participants request documentation. The specific skills available through each varied to some degree, as well. The tutorials and puzzles we used were both designed for

the purpose of teaching programming constructs. Suggestions and documentation both focus more strongly on API methods than constructs. There were a large number of resources of all types: 46 tutorials, 46 puzzles, 68 documentation entries, and 85 suggestions (of which participants were offered an average of 14.25 suggestions ($SD = 6.67$)).

We compared participants' access to and use of learning resources in three different ways: 1) by raw counts, 2) by the proportion of participants who interacted with each type of resource, and 3) by time spent with each resource. We tracked access to each learning resource using the log data showing that resource being opened. We tracked use of skills by filtering log data to find situations in which the participant used a new API or programming construct for the first time after having accessed help related to that topic.

2) Surveys: At the introductory workshop, we asked participants to complete a survey to gather information on demographics and programming experience. Additionally, we wanted to collect information about the rationales behind participants' help usage decisions. To do this, we instrumented Looking Glass to display a quick survey every hour for the first five hours of usage, and every two hours after that. The survey asked participants why they did and did not use particular resources and what they learned from the resources they did use. After completing their film festival submission, we asked participants to complete one final survey asking them how they approached learning skills while creating their submission and how they would advise a friend to use the available resources if preparing for a similar film festival.

3) Complexity: Finally, to enable us to explore the relationship between help access and their final program complexity, we computed a complexity measure. At a high level, we see two classes of skills that programs can incorporate: skills related to API use and skills related to construct use.

While a greater diversity of API use is one reasonable measure to consider, we wanted to augment pure diversity with similarity to the API usage of experts. To do this, we compiled a list of methods used in expert and novice animations sampled from the Looking Glass community. We assigned two points to methods used more by experts than novices and one to methods used more by novices than experts.

We measured the complexity of construct use in a program by summing the nesting degrees for constructs in each participants' film festival submission.

IV. Results

This study seeks to answer four research questions: 1) how much and when did participants access and use each educational resource, 2) why did participants make such learning decisions, 3) how much did the use of educational resources influence the complexity in their final submission, and 4) did demographics affect the usage of educational resources.

A. How much and when did participants access and use each educational resource?

In answering our first research question, we looked at three kinds of data: 1) the numbers of times participants accessed and used methods or constructs from the learning resources, 2) the proportion of users who interacted with each type of learning resource, and 3) the time participants spent with each type of learning resource, see Table I. To reduce the chances of Type 1 error, we used the Bonferroni correction, dividing the cutoff for significance by the number of follow-up tests. This resulted in an adjusted alpha level of 0.0083.

1) Access and Usage Counts by Learning Resource: Participants accessed in-context forms of help more frequently than other learning resources, but used information accessed through suggestions more often. Participants accessed suggestions ($M = 4.2, SD = 5.2$) significantly more often than they accessed tutorials ($M = 0.025, SD = 0.16$) and puzzles ($M = 0.3, SD = 0.88$). There was no significant difference between the number of suggestions and documentation entries ($M = 2.75, SD = 9.15$) participants accessed. We also found that participants used information from suggestions significantly more often than information from any of the other forms of help. Participants used significantly more methods and constructs that they were exposed to via suggestions ($M = 1.275, SD = 1.811$) than through documentation ($M = 0.25, SD = 0.84$), puzzles ($M = 0.325, SD = 0.89$), and tutorials ($M = 0.025, SD = 0.16$). There were no significant differences in the use of information viewed through documentation, tutorials and puzzles. However, both the variation in the learning resources available in different forms and the time investment necessary to use those resources likely have an impact on the raw numbers of accesses.

To address the differences in content availability in resources, we looked at the access and use for the topics available across all four learning resources. Here, we again find a preference to access suggestions ($M = 1.4, SD = 1.6$) over documentation ($M = 0.375, SD = 1.13$), tutorials ($M = 0.025, SD = 0.16$) and puzzles ($M = 0.275, SD = 0.85$). Participants used significantly more methods and constructs after viewing suggestions ($M = 0.275, SD = 0.679$) than after viewing tutorials ($M = 0, SD = 0$), but we did not find a significant difference for the number of methods and constructs viewed through suggestions from those added after documentation ($M = 0.025, SD = 0.158$) or puzzles ($M = 0.125, SD = 0.463$). There were no significant differences in the rates of access and use among documentation, tutorials, and puzzles as well.

2) Proportion of Participants Accessing and Using Learning Resources: To support independent learning, we ideally want a large proportion of users to access learning resources and to use some subset of the methods and constructs introduced by the learning resources they access. Here, we find that suggestions were accessed and used by a significantly larger proportion of participants than the other types of learning resources. 70% of participants accessed at least one suggestion, as compared to 27.5% for documentation, 17.5% for puzzles, and 2.5% for tutorials. Further, 53% of participants used at least one method or construct introduced via suggestions, versus 15% for both documentation and puzzles, and 2.5% for tutorials. The proportion of participants who accessed documentation is also higher than the proportion for tutorials.

TABLE I
SUMMARIZED RESULTS FOR ACCESS AND USAGE OF EDUCATIONAL RESOURCES

Measures	**Set of Topics**	**Main Statistic**	**Paired Post-hoc Tests - p value**					
			Suggest.-Doc.	Suggest.-Tutorial	Suggest.-Puzzle	Doc.-Tut.	Doc.-Puzzle	Tut.-Puzzle
# Accessed	All topics	$F(3,117) = 6.66, p < .001^{***}$	.22	$< .001^{**}$	$.001^{**}$	.02	.04	.82
	Topics in all resources	$F(3,117) = 15.5, p < .001^{***}$	$< .001^{***}$	$< .001^{***}$	$< .001^{***}$	.14	.68	.3
# Used	All topics	$F(3,117) = 10.65, p < .001^{***}$	$< .001^{***}$	$< .001^{***}$	$< .001^{***}$	.36	.76	.22
	Topics in all resources	$F(3,117) = 3.41, p = .02^{*}$	.0084	$.004^{*}$	.11	.79	.29	.18
Use to access ratio	All topics	$F(3,117) = 10.44, p < .001^{***}$	$< .001^{***}$	$< .001^{***}$	$< .001^{***}$	.4	.53	.14
	Topics in all resources	$F(3,117) = 4.11, p = .008^{**}$	$.005^{*}$	$.001^{**}$	.02	0.7	.64	.39
% participants accessed	All topics	$\chi^2(3) = 48.53, p < .001^{***}, \phi_c = .55$	$< .001^{**}$	$< .001^{***}$	$< .001^{***}$	$.005^{*}$	.42	.06
	Topics in all resources	$\chi^2(3) = 38.36, p < .001^{***}, \phi_c = .49$	$< .001^{**}$	$< .001^{***}$	$< .001^{**}$	.11	1	.06
% participants used	All topics	$\chi^2(3) = 33.61, p < .001^{***}, \phi_c = .46$	$< .001^{**}$	$< .001^{***}$	$< .001^{**}$	.11	1	.11
	Topics in all resources	$\chi^2(3) = 11.23, p = .011^{*}, \phi_c = .26$	.06	.02	.31	1	.61	.24
Time spent	All topics	$F(3,117) = 4.21, p = .007^{**}$	$.006^{*}$	$.002^{*}$	.03	.75	.57	.38

Main Statistic - $^{***}p < .001$, $^{**}p < .01$, $^{*}p < .05$

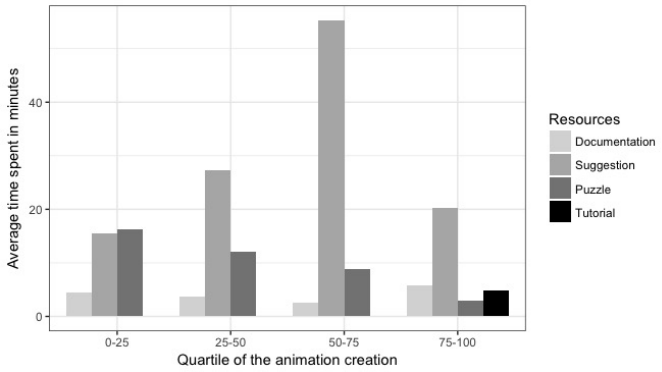

Fig. 2. Amount of time spent (in minutes) on each resource accessed at every quartile of the animation creation

3) Time Spent by Learning Resource: Given that it takes substantially longer to work through tutorials and puzzles than it takes to view a documentation entry or a suggestion, we also compared the time participants spent viewing the different learning resources and how participants allocated that time throughout the study. Participants spent significantly more time viewing suggestions ($M = 2.89, SD = 6.95$) than documentation ($M = 0.41, SD = 1.48$) and tutorials ($M = 0.12, SD = 0.78$). There was no significant difference between the amounts of time participants spent viewing suggestions and puzzles ($M = 0.91, SD = 3.56$).

Participants accessed learning resources throughout the course of the study. Figure 2 shows the total time spent on the four types of learning resources by quartile of time in the study. We determined quartiles based on the total length of time each individual participant worked on his or her film festival submission. Most participants accessed suggestions for an increasing amount of time over the first three quartiles, with a substantial drop during the last quartile. This may suggest that participants found value in the suggestions, as their work patterns changed to incorporate more suggestions later in the session. We think the drop may correspond with participants working to polish their submissions. In contrast, participants' use of documentation and puzzles were highest in the first quartile and decreased in subsequent quartiles. However, we note that a single user paged through more than 30 documentation entries at the end of their creation, causing an increase in the average time spent shown in Figure 2.

B. Why did participants make the above learning decisions?

Overall, the survey results suggest that participants valued the suggestions for opportunistic information and the documentation entries to answer specific questions. Also, most participants expressed a desire to work on their own creations and felt that tutorials and puzzles would take them away from their own work. They perceived the suggestions and documentation as sufficient support for their needs.

1) Opportunistic Information: Participants explained that suggestions often introduced them to new elements to use in their programs. As one participant said, "Tips help suggest things that you might want to include." Another suggested that Looking Glass users should open any offered suggestion "even if it doesn't correspond to the action they wished to complete [because] it would give them ideas." Some participants also valued suggestions that showed them how to use methods in new ways: "I used them because they corresponded with the actions I wished to execute. I learned how to complete the same actions I wished to accept in a more efficient way." Overall, participants seemed to value the fact that suggestions helped to provide the answers to questions they did not know to ask. Some participants reported struggling to use documentation in a similar opportunistic way: commented "I used '?' [i.e. documentation] to see what it would help me with and it didn't help at all."

2) Answering Questions: While suggestions were used more opportunistically, participants seemed to find documentation most helpful in answering a specific question. For

TABLE II
DETAILED STATISTICS FOR ACCESS AND USAGE OF EDUCATIONAL RESOURCES VS. COMPLEXITY

	Model statistic	**Coefficients, t(35) and p values**					
			Intercept	**Suggestion**	**Documentation**	**Tutorial**	**Puzzle**
# Accessed vs. API complexity	$F(4,35) = 9.74, p < 0.001^{***}$ $R^2 = 0.527$	β, t p	13.02, 10.19, $< 0.001^{***}$	0.79, 2.82, 0.008^{**}	0.45, 3.75, $< 0.001^{***}$	−3.73, −0.3, 0.77	−0.44, −0.19, 0.85
# Used vs. API complexity	$F(4,35) = 8.52, p < 0.001^{***}$ $R^2 = 0.493$	β, t p	13.92, 10.49, $< 0.001^{***}$	0.64, 1.03, 0.31	7.12, 5.79 $< 0.001^{***}$	−1.27, −0.19, 0.85	1.17, 0.97, 0.34
# Accessed vs. constructs complexity	$F(3,36) = 0.247, p = 0.86$ $R^2 = 0.02$	β, t p	1.68, 4.44, $< 0.001^{***}$	−0.21, −0.79, 0.44	-	−0.9, −0.23, 0.82	0.21, 0.31, 0.76
# Used vs. constructs complexity	$F(3,36) = 3.68, p = 0.02^{*}$ $R^2 = 0.235$	β, t p	1.12, 3.58, 0.001^{**}	1.54, 3.3, 0.002^{**}	-	−0.45, −0.25, 0.8	0.16, 0.51, 0.61

example, one participant stated "I used '?' help because there was an option that I selected through the 'more' button that I did not understand." However, participants tended to report less success when they could not articulate a specific question. Participants who turned to suggestions with specific questions also struggled: "I used tips to find out how to edit a phrase after I already wrote it. I couldn't find the answer in tips."

3) Remaining in Context: Participants voiced a preference to concentrate on their programs, and felt that tutorials and puzzles would be a diversion. For example, one participant stated "I didn't use the puzzles because I just wanted to focus on my animation". Another said "Tutorials and puzzles might be useful but not if you don't want to exit your program." The tutorials and puzzles inherently ask learners to adopt a new goal that contributes only indirectly to their film festival submission. They may learn a new skill, but will still have to find ways to use that skill in the context of their own program. Participants wanted to continuously feel as though they were making progress on their immediate goals. As one participant stated, "I strongly suggest the tips because they are easy to access and give you very direct directions and examples." However, those that did use puzzles found them both fun and helpful, such as one participant who stated "I mainly just used them to have fun, and I did learn a bit about the coding from them."

Participant surveys also suggest that many participants did not feel a strong need for additional help beyond suggestions and documentation. One participant stated that "I learned about [new topics] from documentation and tips" and, consequently, felt no need to explore the tutorials and puzzles. Others were confident in their skills and abilities overall and did not see utility in tutorials and puzzles, "I didn't use tutorials and puzzles because I have prior experiences with Looking Glass and knew how to code well."

C. How did the use of learning resources relate to the complexity of the submitted animation?

While the access and usage statistics provide some insight into how well learners were able to apply information viewed through the different learning resources, we were also interested in how the access and use of skills from the different learning resources contributed to the overall complexity within participants' final programs. It is important to note that the final complexity measure includes a potentially broader set of API and construct skills than they saw through the learning resources, as some participants may have explored other methods and constructs. To examine the impact of the learning resource types on overall complexity, we constructed a set of four regression models that predicted the construct and method complexity based on the access and usage data for each of the learning resources, see Table II.

TABLE III
ACCESS AND USAGE OF LEARNING RESOURCES BETWEEN DIFFERENT DEMOGRAPHICS

Demographics	**Action for resources**	**Statistics**
Age	Accessed	$F(3,114) = 0.03, p = 0.99$
	Used	$F(3,114) = 0.39, p = 0.76$
Gender	Accessed	$F(3,114) = 0.399, p = 0.75$
	Used	$F(3,114) = 1.095, p = 0.35$
Programming Experience	Accessed	$F(3,114) = 0.17, p = 0.92$
	Used	$F(3,114) = 0.63, p = 0.598$

The regression models for the API complexity explained 52.7% of the variance in API complexity using access and 49.3% of the variance using usage information. Given the amount of individual variation we see among young programmers, we think these are fairly strong models. The coefficients for suggestions and documentation were significant for the access model, but only the documentation coefficient was significant for the usage model. In combination with the access and use results showing less usage of documentation than suggestions, this may suggest that participants who used the documentation were also more likely to explore other API methods, beyond those introduced through documentation.

Our regression models struggle to explain the construct complexity. The access model explains only 2% of variation and is not significant, while usage does somewhat better at 23.5%. Only the suggestions coefficient is significant. This is likely a reflection of participants' tendency to use the suggestions and documentation, overall.

D. Were there differences in usage based on demographics?

Prior studies have found differences in programming behavior by gender [50], [52], age [40], and programming experience [50], [53]–[55]. Because such differences can lead to inequities between groups, we felt it was important to look for

effects regarding the access and usage of learning resources. We found no significant differences in both usage and access of educational resources for age, gender, or programming experience (see Table III). This result contrasts with an earlier, lab-based study that found differences in the use of suggestions and documentation by gender and coding experience [50].

V. Threats to Validity

The primary threat comes from our participant selection. Participants were recruited through a science-related mailing list and may have had stronger motivation towards and aptitude for technical topics and activities. Other groups of youth who do not identify as interested in science may use help resources differently. We also note that the film festival structure created an incentive to keep working on a project. Users who use the system independently, without a specific target in mind may also behave differently.

VI. Discussion and Future Work

Our study is one of the first to explore the usage of different learning resources in a support-rich programming environment. In our study, participants accessed in-context help most frequently because they were strongly motivated to work on their own programs and hesitated to leave their work context to pursue learning goals. They felt that the two in-context help options were sufficient for their needs, although they were used differently. Participants valued suggestions for helping them find new methods and constructs to use or new ways to use ones already in use. In contrast, participants typically used documentation when they had a specific question.

One of the potential concerns around the use of suggestion systems is that not all of the suggestions will be of use to a particular user. This was true in our study. However, participants were 3.7 to 5 times more likely to make use of information from a suggestion than from a documentation entry, based on the different sets of methods evaluated. While not all of the suggestions generated were helpful, overall the system seems to have selected a highly relevant set of content.

While tutorials and puzzles were used less frequently than documentation and suggestions, we note that they actually have the highest use to access ratio: 86% for puzzles and 100% for tutorials. However, the use of puzzles and tutorials were still low compared to the in-context learning resources. We note that the suggestions in particular are designed to help users see their potential relevance within their current program. The suggestions provide descriptions with specific references to objects within users' animation, inviting, for example, a user with an alien in their program to "Make the alien walk faster." Helping users to connect the content in puzzles with their current goals may help to entice more users to complete puzzles and tutorials. One specific opportunity comes through the use of constructs. Construct behavior can be substantially more complex than that of a single API method. Our study suggests that a small population of users may have struggled to learn constructs using suggestions alone. Users who repeatedly open a construct-related suggestion might benefit from a recommendation for a puzzle or tutorial that will help them understand the use of that construct.

Most research to date has focused on the design of out-of-context forms of help, especially tutorials, rather than in-context help, creating an opportunity to further explore in-context help. Our study suggests that users are most likely to explore from in-context forms of help and that suggesting potentially relevant programming elements to explore is of particular value. While suggestions and documentation are two valuable points in the design space of in-context help, we believe these results point to the need for a broader exploration of in-context help for novice programmers.

Further, we note that while the puzzles and tutorials were used less frequently, they resulted in a higher proportion of uses per access than suggestions and documentation. One of the important differences between these two groups of help resources is the time users spend with each help item. Suggestions and documentation are designed for and used quickly. Short usage times may impose limits on what topics users can effectively learn. Another interesting direction for future work is to explore the potential for using in-context help to motivate users to engage with longer-use forms of help such as puzzles and tutorials.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. 1054587 and 1440996.

References

[1] M. B. Rosson, J. Ballin, and J. Rode, "Who, what, and how: A survey of informal and professional web developers," in *Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on*. IEEE, 2005, pp. 199–206.

[2] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer, "Two studies of opportunistic programming: interleaving web foraging, learning, and writing code," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2009, pp. 1589–1598.

[3] B. Dorn and M. Guzdial, "Learning on the job: characterizing the programming knowledge and learning strategies of web designers," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2010, pp. 703–712.

[4] C. R. Rupakheti, D. Hou, and R. Seidel, "Recommending API documentation."

[5] M. P. Robillard, "What makes APIs hard to learn? answers from developers," *IEEE software*, vol. 26, no. 6, 2009.

[6] M. P. Robillard and R. Deline, "A field study of API learning obstacles," *Empirical Software Engineering*, vol. 16, no. 6, pp. 703–732, 2011.

[7] J. Stylos, A. Faulring, Z. Yang, and B. A. Myers, "Improving api documentation using API usage information," in *Visual Languages and Human-Centric Computing, 2009. VL/HCC 2009. IEEE Symposium on*. IEEE, 2009, pp. 119–126.

[8] U. Dekel and J. D. Herbsleb, "Improving API documentation usability with knowledge pushing," in *Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society, 2009, pp. 320–330.

[9] C. Chen and K. Zhang, "Who asked what: Integrating crowdsourced FAQs into API documentation," in *Companion Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 456–459.

[10] C. Treude and M. P. Robillard, "Augmenting API documentation with insights from Stack Overflow," in *Proceedings of the 38th International Conference on Software Engineering*. ACM, 2016, pp. 392–403.

[11] S. Subramanian, L. Inozemtseva, and R. Holmes, "Live API documentation," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 643–652.

[12] D. Hoffman and P. Strooper, "API documentation with executable examples," *Journal of Systems and Software*, vol. 66, no. 2, pp. 143–156, 2003.

[13] R. Holmes, R. J. Walker, and G. C. Murphy, "Approximate structural context matching: An approach to recommend relevant examples," *IEEE Transactions on Software Engineering*, vol. 32, no. 12, 2006.

[14] M. Gasparic, A. Janes, F. Ricci, and M. Zanellati, "GUI design for IDE command recommendations," in *Proceedings of the 22nd International Conference on Intelligent User Interfaces*. ACM, 2017, pp. 595–599.

[15] J. Brandt, M. Dontcheva, M. Weskamp, and S. R. Klemmer, "Example-centric programming: integrating web search into the development environment," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2010, pp. 513–522.

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

[17] S. Dasgupta, W. Hale, A. Monroy-Hernández, and B. M. Hill, "Remixing as a pathway to computational thinking," in *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*. ACM, 2016, pp. 1438–1449.

[18] P. Brusilovsky, "Webex: Learning from examples in a programming course." in *WebNet*, vol. 1, 2001, pp. 124–129.

[19] R. Hosseini, T. Sirkiä, J. Guerra, P. Brusilovsky, and L. Malmi, "Animated examples as practice content in a java programming course," in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. ACM, 2016, pp. 540–545.

[20] W. Jernigan, A. Horvath, M. Lee, M. Burnett, T. Cuilty, S. Kuttal, A. Peters, I. Kwan, F. Bahmani, and A. Ko, "A principled evaluation for a principled Idea Garden," in *Visual Languages and Human-Centric Computing (VL/HCC), 2015 IEEE Symposium on*. IEEE, 2015, pp. 235–243.

[21] M. Ichinco, W. Hnin, and C. Kelleher, "Suggesting examples to novice programmers in an open-ended context with the Example Guru," in *Visual Languages and Human-Centric Computing (VL/HCC), 2016 IEEE Symposium on*. IEEE, 2016, pp. 230–231.

[22] A. Davidovic, J. Warren, and E. Trichina, "Learning benefits of structural example-based adaptive tutoring systems," *IEEE Transactions on Education*, vol. 46, no. 2, pp. 241–251, 2003.

[23] B. B. Morrison, L. E. Margulieux, and M. Guzdial, "Subgoals, context, and worked examples in learning computing problem solving," in *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*. ACM, 2015, pp. 21–29.

[24] P. J. Guo, "Online Python Tutor: Embeddable web-based program visualization for cs education," in *Proceeding of the 44th ACM technical symposium on Computer science education*. ACM, 2013, pp. 579–584.

[25] A. Head, C. Appachu, M. A. Hearst, and B. Hartmann, "Tutorons: Generating context-relevant, on-demand explanations and demonstrations of online code," in *Visual Languages and Human-Centric Computing (VL/HCC), 2015 IEEE Symposium on*. IEEE, 2015, pp. 3–12.

[26] P.-Y. Chi, S. Ahn, A. Ren, M. Dontcheva, W. Li, and B. Hartmann, "MixT: automatic generation of step-by-step mixed media tutorials," in *Proceedings of the 25th annual ACM symposium on User interface software and technology*. ACM, 2012, pp. 93–102.

[27] K. Knabe, "Apple guide: a case study in user-aided design of online help," in *Conference companion on Human factors in computing systems*. ACM, 1995, pp. 286–287.

[28] S. Palmiter and J. Elkerton, "Animated demonstrations for learning procedural computer-based tasks," *Human-Computer Interaction*, vol. 8, no. 3, pp. 193–216, 1993.

[29] C. Kelleher and R. Pausch, "Stencils-based tutorials: design and evaluation," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 2005, pp. 541–550.

[30] J. Fernquist, T. Grossman, and G. Fitzmaurice, "Sketch-sketch revolution: an engaging tutorial system for guided sketching and application learning," in *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, 2011, pp. 373–382.

[31] S. Pongnumkul, M. Dontcheva, W. Li, J. Wang, L. Bourdev, S. Avidan, and M. F. Cohen, "Pause-and-play: automatically linking screencast video tutorials with applications," in *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, 2011, pp. 135–144.

[32] T. Grossman and G. Fitzmaurice, "ToolClips: an investigation of contextual video assistance for functionality understanding," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2010, pp. 1515–1524.

[33] B. Lafreniere, T. Grossman, and G. Fitzmaurice, "Community enhanced tutorials: improving tutorials with multiple demonstrations," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2013, pp. 1779–1788.

[34] J. Matejka, T. Grossman, and G. Fitzmaurice, "Ambient Help," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2011, pp. 2751–2760.

[35] D. Parsons and P. Haden, "Parson's programming puzzles: a fun and effective learning tool for first programming courses," in *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52*. Australian Computer Society, Inc., 2006, pp. 157–163.

[36] T. Sirkiä, "Combining Parson's problems with program visualization in CS1 context," in *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*. ACM, 2016, pp. 155–159.

[37] B. J. Ericson, "Dynamically adaptive parsons problems," in *Proceedings of the 2016 ACM Conference on International Computing Education Research*. ACM, 2016, pp. 269–270.

[38] "Hour of Code." [Online]. Available: https://code.org/

[39] K. J. Harms, N. Rowlett, and C. Kelleher, "Enabling independent learning of programming concepts through programming completion puzzles," in *Visual Languages and Human-Centric Computing (VL/HCC), 2015 IEEE Symposium on*. IEEE, 2015, pp. 271–279.

[40] K. J. Harms, J. Chen, and C. L. Kelleher, "Distractors in Parson's problems decrease learning efficiency for young novice programmers," in *Proceedings of the 2016 ACM Conference on International Computing Education Research. ACM*, 2016, pp. 241–250.

[41] H. Li, Z. Xing, X. Peng, and W. Zhao, "What help do developers seek, when and how?" in *Reverse Engineering (WCRE), 2013 20th Working Conference on*. IEEE, 2013, pp. 142–151.

[42] E. Duala-Ekoko and M. P. Robillard, "Asking and answering questions about unfamiliar APIs: An exploratory study," in *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press, 2012, pp. 266–276.

[43] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung, "An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks," *IEEE Transactions on software engineering*, vol. 32, no. 12, 2006.

[44] "Looking Glass." [Online]. Available: http://lookingglass.wustl.edu/

[45] "Scratch." [Online]. Available: https://scratch.mit.edu/

[46] M. J. Conway, *Alice: Easy-to-learn three-dimensional scripting for novices*, 1998.

[47] M. Kölling, "The Greenfoot programming environment," *ACM Transactions on Computing Education (TOCE)*, vol. 10, no. 4, p. 14, 2010.

[48] K. J. Harms, E. Balzuweit, J. Chen, and C. Kelleher, "Learning programming from tutorials and code puzzles: Children's perceptions of value," in *Visual Languages and Human-Centric Computing (VL/HCC), 2016 IEEE Symposium on*. IEEE, 2016, pp. 59–67.

[49] M. J. Lee, "Gidget: An online debugging game for learning and engagement in computing education," in *Visual Languages and Human-Centric Computing (VL/HCC), 2014 IEEE Symposium on*. IEEE, 2014, pp. 193–194.

[50] M. Ichinco, W. Hnin, and C. Kelleher, "Suggesting API usage to novice programmers with the Example Guru," 2017.

[51] T. Thomas, B. Chu, H. Lipford, J. Smith, and E. Murphy-Hill, "A study of interactive code annotation for access control vulnerabilities," in *Visual Languages and Human-Centric Computing (VL/HCC), 2015 IEEE Symposium on*. IEEE, 2015, pp. 73–77.

[52] L. Beckwith, C. Kissinger, M. Burnett, S. Wiedenbeck, J. Lawrance, A. Blackwell, and C. Cook, "Tinkering and gender in end-user programmers' debugging," in *Proceedings of the SIGCHI conference on Human Factors in computing systems*. ACM, 2006, pp. 231–240.

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

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

[55] J. N. Matias, S. Dasgupta, and B. M. Hill, "Skill progression in scratch revisited," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 2016, pp. 1486–1490.