From Barriers to Learning in the Idea Garden: An Empirical Study

Jill Cao[1], Irwin Kwan[1], Rachel White[1], Scott D. Fleming[2], Margaret Burnett[1], Christopher Scaffidi[1]

[1]School of Electrical Engineering and Computer Science
Oregon State University
Corvallis, U.S.A.
{caoch, kwan, white, burnett, cscaffid}@eecs.oregonstate.edu

[2]Department of Computer Science
University of Memphis
Memphis, U.S.A.
Scott.Fleming@memphis.edu

***Abstract*—How can end-user programming environments better help their users overcome programming barriers? We have been investigating an approach called Idea Gardening, which addresses this problem by helping end users to help themselves overcome barriers in the context of "doing". In this paper, we report on a qualitative empirical study of how effectively an Idea Garden prototype helped end users overcome programming barriers in the CoScripter environment, and the extent to which participants learned after interacting with our features. Our results showed that 9 out of 10 participants who encountered barriers and then used the Idea Garden, overcame their barriers. Further, all 9 went on to demonstrate evidence of having learned the programming concepts, patterns, and strategies relevant to overcoming these barriers.**

Keywords - end-user programming; Idea Garden; learning

I. INTRODUCTION

For many end users, programming is a challenging task that requires overcoming numerous barriers, such as decomposing design problems [7], using primitives such as loops [9], and selecting and combining modules [24]. Although recent work has made advances toward alleviating such difficulties (e.g., [14][26]), empirical studies show that end-user programmers continue to encounter barriers when creating or maintaining programs in a wide range of programming environments, including spreadsheets [12][23], animations [18][27], mashups [7][8][9][33], and Visual Basic [24].

One reason barriers persist may be that there is little support for helping end-user programmers gain the *problem-solving skills* needed to overcome barriers they encounter. Tutorials and training materials on programming (e.g., [19][21]) have been proposed to address this problem. However, Minimalist Learning Theory [10] suggests that task-oriented users will resist investing time on such learning materials, preferring to pick up the knowledge they require in the context of their tasks.

To fill this gap, we previously proposed the Idea Garden approach [9]. Consistent with Minimalist Learning Theory, the Idea Garden seeks to balance learning and task orientation by *integrating learning into the user's own programming tasks*. Further, it aims to help end-user programmers learn to solve problems *for themselves* (as opposed to aiming to automatically solve users' problems for them). Thus, the Idea Garden equips a programming environment with features that facilitate learning how to solve barriers in the user's own tasks.

To investigate whether the Idea Garden can deliver these benefits, we prototyped the Idea Garden within the CoScripter end-user web-scripting environment. For this environment, the Idea Gardening features target two barriers: (1) Composition, the inability to compose existing functionality, and (2) More-Than-Once, the inability to generalize calculations on a single data item to multiple data [8][9]. Our features aim to help users overcome these two barriers by conveying two programming concepts (Iteration and Dataflow), two patterns (Webpage-as-Component and Repeat-Copy-Paste) and two problem-solving strategies (Analogy and Generalization).

Thus, two overall questions arise: whether the Idea Garden helps end users to overcome barriers in their programming tasks, and if so, whether any learning happens along the way. In this paper, we investigate these questions through a qualitative study in which end users interacted with the Idea Garden in the context of a programming task. For our study, we refined the above two overall questions into the following three specific research questions:

- RQ1: Does the Idea Garden help users complete programming tasks?
- RQ2: Which barriers does the Idea Garden help users overcome?
- RQ3: Do users who use the Idea Garden when they encounter barriers then learn relevant programming concepts, patterns, and/or problem-solving strategies?

II. BACKGROUND AND RELATED WORK

A. Helping Users with Programming Barriers

Empirical studies have shown that end-user programmers face numerous barriers to succeed in programming tasks, such as difficulties decomposing design problems, selecting APIs, combining APIs, and problem-solving in their programming environments. Studies report these difficulties when users create or maintain programs such as spreadsheets [12][23], animations [18][27], and mashups [7][9][8][33].

One approach is for tools to try to automatically remove these barriers. For example, programming-by-demonstration tools allow the end-user programmer to avoid the need to think about language constructs, variables, etc., and to instead show concrete examples of the desired program behavior, from which the tools automatically infer the program [14]. Other tools aim to automatically correct errors, enabling the user to

978-1-4673-0853-3/12/$31.00 ©2012 IEEE

enter a mostly correct program from which the tools automatically infer the desired behavior [27]. Still other tools simplify programming by providing templates that describe classes of nearly finished programs that users complete by simply filling in the blanks (e.g., [32]). A similar approach is for tools to provide examples from a set of recommendations that users can copy, paste, and tweak (e.g., [20]). In fact, some tools offer a menu of code snippets that can be automatically generated on demand (e.g., [16]). User studies in the papers cited above demonstrate that by simplifying the programming task, these and similar tools help people to complete programming tasks more quickly with fewer errors.

However, automatically removing barriers comes with a cost: the users do not need to learn how to overcome barriers by themselves because they can instead simply rely on the tool to automatically provide a solution. In fact, some novice programmers consciously opt not to learn some skills, believing that they can always crib from the web [4]. But since no tool can automatically solve every programming difficulty, some amount of learning about how to solve programming problems seems important, even for end-user programmers.

Some approaches do emphasize helping users to learn. For example, tutorials and training materials explicitly aim to help people who want to develop their programming skills. Such approaches have shown promise in helping novice programmers to learn skills in classes or in online communities (e.g., [19][21]). However, some users are so focused on a task at hand that they are unwilling to invest time in taking tutorials, reading documentation, or using other training materials—even if such an investment might be rational in the long term. This phenomenon is known as the *paradox of the active user* [11].

Our work seeks to strike a balance between these two extremes by integrating learning into the context of users' programming tasks in a manner consistent with Minimalist Learning Theory [10]. A few other approaches also seek to integrate learning into programming. One approach, explored in the context of animation, had novice programmers create basic programs, such as 2D animations, and then provided features enabling the users to convert their animations to 3D [22]. The approach included a curriculum for teachers to help users choose features for overcoming barriers. Studies showed that this educational scaffolding helped programmers learn these features and improve their 3D programming. Another approach, explored in the context of animations and mashups, sought to help programmers learn by downloading and extending other users' programs [25][29]. These tools have been shown to increase programmers' ability to learn from existing programs [25].

The main differences of our Idea Garden approach from these prior works are that the Idea Garden integrates scaffolding for learning *directly into the programming tool itself*. Thus, the approach aims to support integrated learning when no teacher or relevant programs are available. In this approach, the chief design challenges are (1) how to avoid automatically solving the problem for the user, (2) how to contextualize learning within the programming task, and (3) how to avoid the requirements of human teachers or respositories of suitable example programs.

B. *The Idea Garden's Host: CoScripter*

To facilitate investigation into whether the Idea Garden approach can help end-user programmers overcome barriers and acquire programming knowledge, we implemented the Idea Garden within CoScripter/Vegemite [26], an end-user programming-by-demonstration environment for web automation in Firefox. A formative study of CoScripter [9] showed that end users programming in CoScripter encountered many of the barriers reported in other end-user programming environments (e.g., [24])—barriers such as how to coordinate and compose modules and how to iterate over data.

Using CoScripter, a user can demonstrate how to carry out a task in Firefox. CoScripter translates the user's actions into a "web macro" script that the user can edit and execute (Figure 1a). CoScripter provides a scratch table (Figure 1b) that makes it possible to create mashups that combine data from multiple web pages. For example, a user can create a script to mash restaurant location with public transit by loading a web page of restaurants (Figure 1c), copying its addresses to the table (Figure 1b), then iterating to send each address to another web page to compute travel time via transit. Thus, CoScripter has programming concepts such as control flow and dataflow.

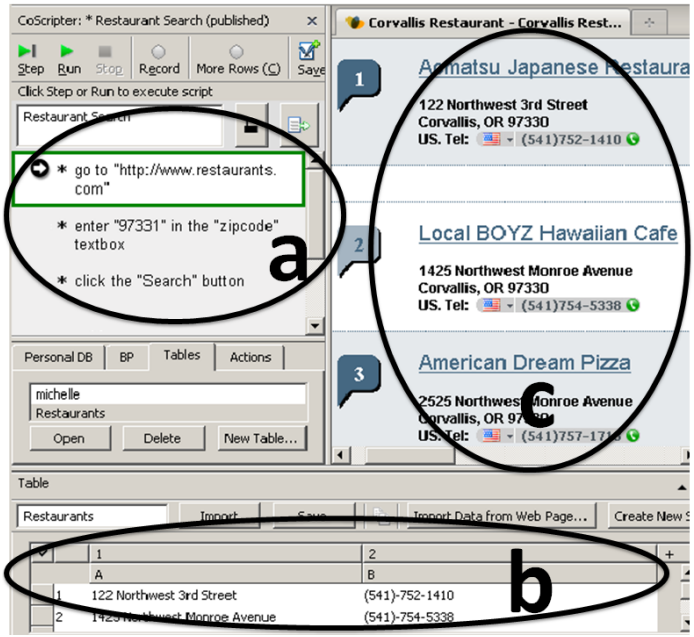

Figure 1. CoScripter's (a) script area, (b) table area, and (c) browsing area.

III. THE STUDY

To investigate how well the Idea Garden helps end users overcome programming barriers and learn, we conducted a qualitative think-aloud study. Qualitative empirical studies involve the collection and analysis of qualitative data, such as verbal protocols, as opposed to quantitative data, such as numeric measurements [31]. Qualitative studies are particularly appropriate for studying complex human behaviors, such as our interest in problem solving by end-user programmers [31].

A. Participants

We used emails and flyers to recruit 15 university students and recent graduates with majors other than electrical engineering or computer science. None had ever programmed before. Of the 15, 10 used the Idea Garden's features (6 females, 4 males; gender is indicated as F or M in IDs below).

B. Study Environment: The Idea Garden Prototype

Our study focused on two barriers identified in earlier work [8][9]: the *Composition* barrier and the *More-Than-Once* barrier. The Composition barrier occurs when a user cannot identify how to combine functionality of existing modules. For example, users may struggle to combine data from multiple webpages. The More-Than-Once barrier occurs when a user cannot generalize calculations on one data item to many data items. An example of this barrier in CoScripter is a user not knowing how to repeat actions on cells in a column. Three features of the Idea Garden target these barriers (Figure 2).

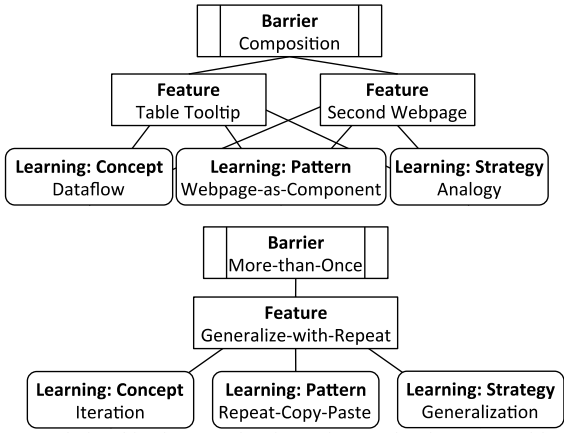

Figure 2. Relationships between barriers, Idea Garden features, and learning objectives.

For example, the *Second Webpage* feature (Figure 3) aims to help users apply the *Webpage-as-Component* pattern and the *Dataflow* concept to overcome the Composition barrier. The Webpage-as-Component pattern uses certain webpages, such as Google Maps, to calculate values from a given input. The Dataflow concept refers to moving data from one webpage to another. The Second Webpage feature draws from this pattern and concept: it explains that the user can use a second webpage to calculate new data, gives an example, and describes how a user can pass data from one webpage to another. Users can access this feature through a "Help" button.

The *Table Tooltip* feature (Figure 4) also seeks to address the Composition barrier. It appears when the user hovers the mouse over the head of a column that has been labeled or populated with data. The feature contextualizes suggestions based on the column's content type (e.g., names, addresses, currency). This feature introduces the Webpage-as-Component pattern by suggesting appropriate pages that the user can visit based on the current data type in the table, and it reinforces the Dataflow concept by suggesting that the user send the data currently in the table to the recommended webpage.

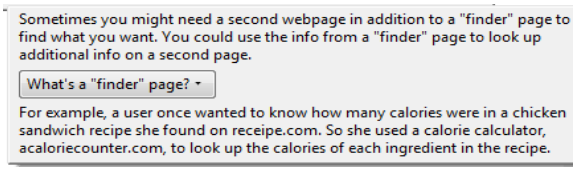

Figure 3: The Second Webpage feature targets the Composition barrier using the Webpage-as-Component pattern, Dataflow concept, and Analogy strategy.

By design, the Second Webpage and the Table Tooltip features suggest solutions that are not necessarily correct. For example, the Second Webpage feature presents a static example of how to calculate calories for a recipe, which is unlikely to be what the user wants. By giving incorrect suggestions, the features aim to entice the user to adopt the *Analogy* strategy—that is, to relate an understood problem (the example in each suggestion) to the problem at hand (the task) [28].

To address the More-Than-Once barrier, the Idea Garden provides the *Generalize-with-Repeat* feature, available via an icon inserted beneath scripts that operate on tables (Figure 5). This feature describes the *Iteration* concept and provides an example snippet of script that uses the `repeat` command. The feature's suggestion also encourages the user to apply the *Generalization* strategy, where a person extends his or her consideration of one object (e.g., a table cell) to the consideration of a set of objects (e.g., a column of cells) [28]. The feature thus demonstrates the *Repeat-Copy-Paste* pattern, which solves the problem of passing multiple entries from the table to a webpage for further processing. It loads the calculation/lookup webpage in the browser, copies/pastes data from the table to the webpage, and then does a calculation or lookup. We envision this pattern and the Webpage-as-Component pattern belonging to a pattern catalog for web users, such as in [15].

C. Study Design

We gave each participant a background questionnaire that included a standard computer self-efficacy test [13]. Although one of our research questions focuses on learning, we chose not to use a pre-test of programming knowledge because doing so could have biased behavior—for example, causing users to focus on features that seemed related to the pre-test. Instead, we used temporal evidence (described later) to detect learning.

Because the Idea Garden is intended for users who have used an environment before (but then get stuck), we gave a 25-minute hands-on tutorial then walked participants through how to create three scripts: one to look up information from a webpage, one to pull data from a webpage into a table, and one to push data from the table to a webpage. We taught some concepts and patterns, but no strategies. Specifically, we taught the Dataflow and Iteration concepts, and had participants do all of the Repeat-Copy-Paste pattern and part of the Webpage-as-

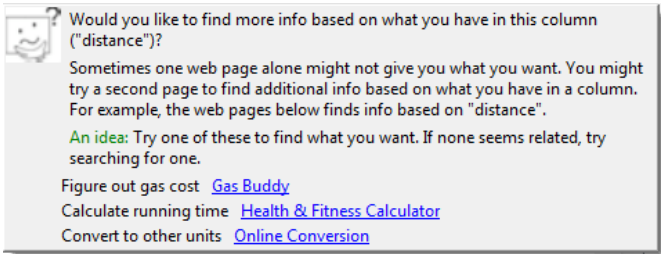

Figure 4: The Table Tooltip feature offers a suggestion ("Sometimes...") and gives example websites the user can explore.

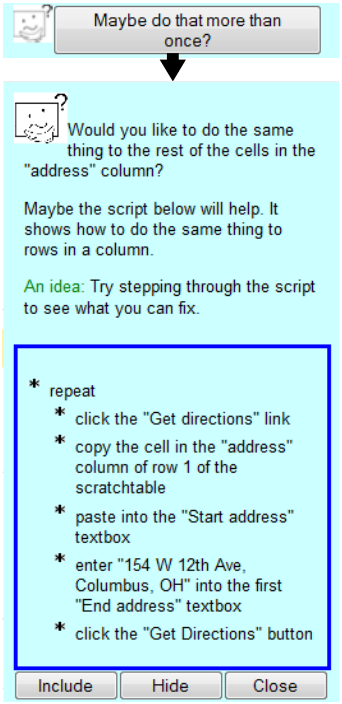

Figure 5: The Generalize-with-Repeat feature suggests the use of the `repeat` construct in CoScripter.

Component pattern. We did not teach strategies per se, although it may have been possible for some participants to infer strategies from their tutorial work. The tutorial's goal was to give participants a basic level of CoScripter ability and practice, but we did not include the Idea Garden in the tutorial. Our intent was that participants should already know enough about CoScripter to start a task, but to use the Idea Garden (if they so chose) without having used it before.

Next, participants began their main task: to create a script for finding 2-bedroom apartments under $1300 within a 10-minute drive of the Ohio State University campus. The task had three implicit subtasks: (1) import a list of apartments and their addresses from a webpage into the table, (2) iterate over the addresses to compute driving time to Ohio State University, and (3) copy each driving time back to the table. Participants were vulnerable to the Composition barrier during subtasks 1 and 2 and to the More-Than-Once barrier during subtasks 2 and 3. The Idea Garden was active during the task, and users could refer back to the tutorial.

If a participant became so stuck on a barrier that no further progress seemed possible, the researcher pointed out Idea Garden features or, if that did not help, eventually suggested an action to overcome the barrier. These hints enabled the participant to make progress, so that we could gather data on later subtasks. To support analysis, participants were instructed to talk aloud as they worked. We recorded audio, screen captures, and video of participants.

Once participants finished the task or exceeded 55 minutes, we used a structured interview to assess knowledge of each programming concept, pattern, and problem-solving strategy in the Idea Garden's learning objectives (Figure 2). Bloom's taxonomy identifies six levels of knowledge: remembering, understanding, applying, analyzing, evaluating, and creating knowledge [1]. Because the task was short, our interview targeted the three lowest levels of this taxonomy. When structuring questions, we focused on learning transferability to another task or context [5]; specifically, we presented participants with the hypothetical task of creating a script to look up the best price for a novel sold online, then asked how to perform subtasks in that context. To further evaluate understanding, we used multiple choice and fill-in-the-blank questions, with follow-up questions asking participants to interpret various portions of scripts and to predict script behaviors.

IV. Analysis Method

Two researchers independently used qualitative thematic coding to analyze the videos. They coded (1) *barriers encountered* based on whether participants struggled in specified ways and (2) *progress made* including *barriers overcome* (Table I). They also graded interviews' multiple choice and fill-in-the-blank answers using an answer key. Finally, participant interviews' task answers were categorized using the concept, pattern, and strategy names in Figure 2. To ensure reliability, we used a standard inter-rater agreement exercise [31] in which, first, the independent analyses of the two coders achieved 80% agreement (Jaccard similarity) on 30% of the data, and then one of the coders alone completed the analysis.

TABLE I. CODING SCHEME FOR WHETHER (1) A PARTICIPANT ENCOUNTERED A BARRIER AND (2) PARTICIPANT MADE PROGRESS.

1. Barrier	**Action/Vocalization**
Composition Barrier	Not knowing to combine *multiple* web pages
	Not knowing that a page *can calculate* driving time
	Not knowing *how* to use a page to calculate
	Not using the table as intermediate storage for webpage data that must be sent to a second webpage
More-Than-Once Barrier	Not realizing operations on table rows could be generalized using the `repeat` command
	Uncertainty about how to use or the misuse of the `repeat` command, e.g. wrong placement or syntax
	Misunderstanding of the scope of the `repeat` command, e.g., thinking it could work on elements of a webpage
	Revisiting tutorial script for help with the `repeat` command
2. Progress	**Observed Behavior**
No movement	Behavior remained unchanged
Movement	Behavior changed but did not move closer to completing task
Positive Movement	Behavior changed and moved closer to completing task
Barrier Overcame	Behavior changed to overcame the barrier

V. Results

A. RQ1: Idea Garden's Help with the Task

Almost all participants who turned to the Idea Garden for help with a barrier were able to complete at least part of the task. For the 10 participants who turned to the Idea Garden when they encountered a barrier, four went on to complete all three subtasks, and four more completed at least one subtask. Table II shows the details.

B. RQ2: Idea Garden's Help with the Barriers

As explained in Section III.B, the Table Tooltip and the Second Webpage features target the Composition barrier, whereas the Generalize-with-Repeat feature targets the More-Than-Once barrier. We will consider these features in the context of the barriers they target.

1) Overcoming the Composition Barrier

Table III summarizes, for the six participants who encountered the Composition barrier and then interacted with the Idea Garden, their feature usage and how successful they were in overcoming the barrier. As the table shows, half the participants who encountered the barrier were able to overcome it.

With the split in participants' success in overcoming the Composition barrier, we analyzed whether these participants exhibited behavior consistent with an understanding of the

TABLE II. PARTICIPANT'S PROGRESS IN THE TASK USING IDEA GARDEN.
● BARRIER, THEN PARTICIPANT USED IDEA GARDEN TO OVERCOME BARRIER.
⊖ BARRIER, THEN PARTICIPANT USED IDEA GARDEN TO CHANGE APPROACH (BUT DID NOT OVERCOME BARRIER).
○ BARRIER, THEN PARTICIPANT USED IDEA GARDEN, BUT DID NOT ACT ON IT.
✔ PARTICIPANT COMPLETED SUBTASK.
✘: PARTICIPANT STARTED BUT DID NOT FINISH THE SUBTASK.
R: RESEARCHER SOLVED THE SUBTASK.
✔R: RESEARCHER COMPLETED PART, PARTICIPANT COMPLETED THE REST.
✘R: PARTICIPANT STARTED BUT DID NOT FINISH THE SUBTASK AND RELIED ON RESEARCHER TO SOLVE THE PART OF THE SUBTASK HE STARTED.

		F1	F2	F3	M1	M2	F4	M3	F5	M4	F6
Sub-task 1	Used Idea Garden	●		⊖	○	○	●	●			
	Task Success	✔	✔	R	R	R	✔	✔	✔	✔	✔
Sub-task 2	Used Idea Garden	●	●	●	○	●		○	●	●	●
	Task Success	✔	✔	✔R	✘R	✔	✔	✔	✔	✔	✔
Sub-task 3	Used Idea Garden								●	●	●
	Task Success		✘	✘			✔	✘	✔	✔	✔
Task Success Total		2/2	2/3	0.5/3	0/2	1/2	3/3	2/3	3/3	3/3	3/3

Webpage-as-Component pattern, Analogy strategy, and Dataflow concept that the Table Tooltip and Second Webpage aimed to convey. Below, we present episodes that shed light on whether participants picked up this knowledge or not, and why.

Applying the Analogy strategy. F4's interactions with the Table Tooltip feature show how she successfully used the Analogy strategy to figure out how to compute driving time. Figure 6 illustrates her interactions with CoScripter and the Idea Garden. The Analogy strategy rests on the ability to effectively map concrete examples to the task at hand. At first F4 tried to find a web page that showed the needed data (driving times), rather than computing the values with a calculator web page (Google Maps). She could not find a suitable site, but added a "distance" column header to her table anyway. This new column head included an indicator for the Table Tooltip feature. The tooltip suggested websites that accepted the data in her column (distance) as input to calculate new information. These websites included a gas calculator and a running time calculator, neither of which was what F4 needed. After inspecting each site, she searched for an analogous website that was appropriate for her task, a "driving distance calculator." Her search led her to a page that she used to finish Subtask 1.

In contrast to F4, neither F3 nor M2 sought websites analogous to the ones provided by the Table Tooltip feature. Both F3 and M2 dismissed the Table Tooltip and went back to their prior approach of looking for driving time from an apartment-listing page. Similarly, M1 interacted with the Second Webpage feature, which showed an example of using a calculator webpage to compute calories of a recipe; however, he appeared not to even read the example, and thus, he lost the opportunity to apply an analogy.

Applying the Webpage-as-Component pattern and the Dataflow concept. Three participants used the Idea Garden's suggestion directly to apply the Webpage-as-Component pattern and the Dataflow concept. In F4's episode (above), she applied the Webpage-as-Component pattern, using a travel calculator page as a component. In leveraging the page, she also applied the Dataflow concept, pushing addresses into the component

TABLE III. THE SIX PARTICIPANTS WHO ENCOUNTERED THE COMPOSTION BARRIER AND THEIR FEATURE USAGE.
● PARTICIPANT USED THE FEATURE AND OVERCAME A BARRIER.
● PARTICIPANT OBSERVED THE FEATURE, MADE MOVEMENT.
○ PARTICIPANT OBSERVED THE FEATURE, BUT MADE NO MOVEMENT.

Feature	Participants who encountered Composition barrier					
	F1	F3	M1	M2	F4	M3
Table Tooltip	●	○○		○	●	
Second Webpage		○●	○	○○		●

and pulling driving times out. Similarly, F1 used Bing Maps, which was suggested by the Table Tooltip, to compute driving times, thus applying the Webpage-as-Component pattern and Dataflow concept. For M3, the Second Webpage feature brought to his attention that he could send existing data to a calculator page to get driving time, thus he too applied the Webpage-as-Component pattern and Dataflow concept.

In contrast, F3, M1, and M2 exhibited no evidence of understanding or applying the Webpage-as-Component Pattern and the Dataflow concept. Even after the researcher directed these participants to an appropriate component page, Google Maps, they continued to struggle. For example, M2 used Google Maps to find apartments rather than to compute driving times. The researcher provided each participant with one more hint: use "Get Directions". But the hint led to more confusion. For example, M1 said: *"I'm just looking for the location of one place, not directions from me to it"*.

These participants' difficulties may be understood in terms of *reframing*, or more specifically, a lack of it. According to Schön [30], a *frame* is a boundary within which people work to solve a messy problem. The participants seemed to frame the task as one of using a web page as a static source of information rather than a component in a computation. These web pages do not present themselves as components in computations since it is not their primary function, so, it is perhaps not surprising that these participants did not break out of their unworkable frame. F3 attempted unsuccessfully to reframe after seeing the Second Webpage feature. She reacted to the feature's suggestion by opening Google Maps; however, she did not see how to leverage the site as a component. Thus, even though the Idea Garden feature did not help F3 overcome the Composition barrier, it did bring her to reflect on and attempt to adjust her frame.

2) *Overcoming the More-Than-Once Barrier*

The More-Than-Once barrier is the target of the Generalize-with-Repeat feature—and 7 of the 9 participants who turned to this feature overcame the barrier and completed the second subtask (Table IV). Because of this success, we consider how the feature might have influenced different participants' behaviors.

Explanation content: Recall from Figure 5 that the explanation briefly gave a general idea and then presented an incomplete script snippet that participants could include in their scripts and then modify. Our intent was that users would include it, notice the missing piece, i.e., the code to pull drive times to the table, then fill it in. And for F5, M4, F6, the explanation led them to do exactly that. (F2 also noticed the missing piece but ran out of time to act upon it.) Figure 7 illustrates the way this played out for M4. The explanation helped F1 in an-

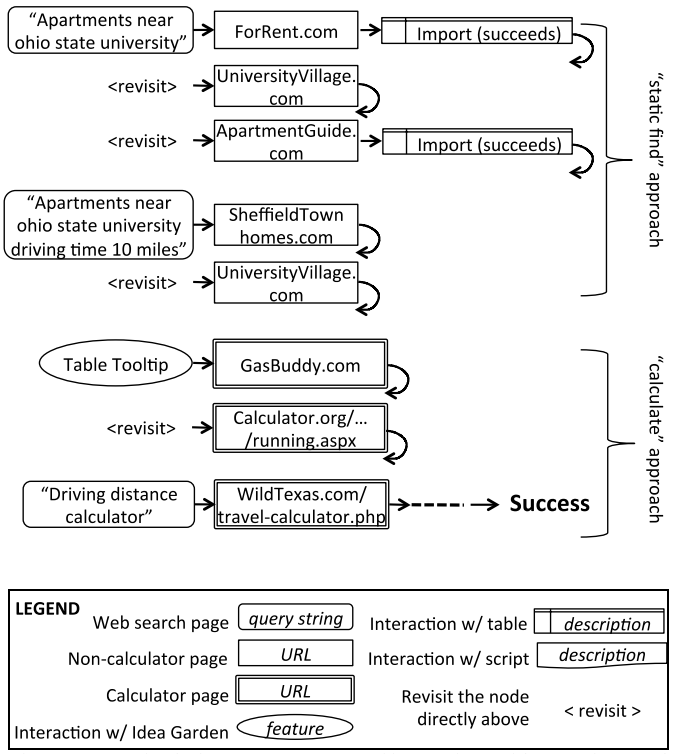

Figure 6. An episode from F4's sequence of interactions with CoScripter and the Idea Garden. (Curved arrows indicate transition to the next line).

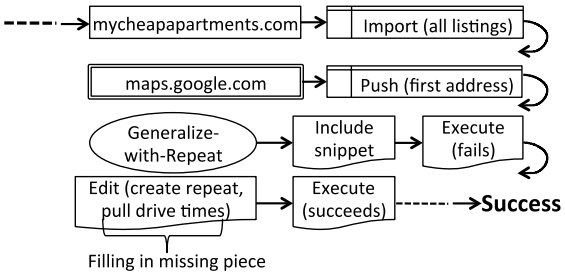

Figure 7. An episode from M4's session (Legend in Figure 6).

other way: she remembered the `repeat` command but had forgotten how to do it. She then used the explanation to remember: *"I forgot how to repeat ... [Opens the suggestion] Ah, repeat... [Reads suggestion line by line and begins to edit a Repeat loop into her script]."*

Two other cases of content were different, in that they were related to recognizing the *relevance* of repetition rather than *how* to implement repetition. In M2's case, he had not realized that his actions were generalizable at all. Instead, he processed the first three rows of his table one at a time. When he noticed the explanation, he used the script snippet it suggested, and changed to a loop rather than his previous one-at-a-time approach: *"[reads: Would you like to do the same thing to the rest of the cells?] Oh okay [reads the suggestion's script, then clicks "Include" to insert it into his script.]"* Finally, M1's case showed a need for explanation content that was not present. There is a "role expressiveness" issue [17] in the way a single entry of the `repeat` body appears in CoScripter scripts: the notation refers to the first row only of the table (see the

TABLE IV. THE NINE PARTICIPANTS WHO ENCOUNTERED THE MORE-THAN-ONCE BARRIER. DOT NOTATION IS AS IN TABLE III.

Participants who used the Generalize-with-Repeat feature								
F1	F2	F3	M1	M2	M3	F5	M4	F6
●	●	●	○	●	○	●	●	●
Content Timing	Content Timing Self-Eff	Timing	Content	Content Timing Self-Eff	Self-Eff	Content Timing Self-Eff	Content Timing	Content Timing

script in Figure 5). This notation convinced M1 that `repeat` was not relevant, so he did not follow the suggestion. If the explanation content had clarified this issue, perhaps he would have.

Timing and context: A strength of the Generalize-with-Repeat feature seems to be that it appears in the right context: when the user is actively working on a portion of the script that involves a row of the table, but is not using a `repeat` in that portion of the script. Further, the "Include" button automatically places the snippet into the user's script, reducing the cost of integrating the snippet, and encouraging users to start experimenting with it. This contrasts with a tutorial, which appears in a separate context and provides no direct encouragement or explicit support for integration with the task at hand. In fact, all seven participants who encountered the More-Than-Once barrier and managed to accomplish the second subtask, did so by using the feature as a reference to implement their own `repeat` loop or by including the script snippet it provided.

The experience of M4 illustrates the value of integrating suggestions into the context of the task at hand. When he encountered the More-Than-Once barrier, M4 referred to a tutorial example that illustrated the use of the `repeat` command. In response, however, he made no changes to his script. Instead, he subsequently tried copying an address from the table into Google Maps. This triggered the Generalize-with-Repeat feature. In contrast to his non-use of the tutorial, he incorporated the suggested snippet, then stepped through it in the context of his own program. After finding where further work was needed, he was able to create a new repeat block that effectively completed the second subtask without any further assistance. This timing and context seemed very effective, and may be the primary reason so many of the participants acted upon the suggestion in a way that turned out well for their scripts.

Self-efficacy: Self-efficacy is a specific form of self-confidence: a person's prediction of how well he or she can perform a specific task [2]. Results from empirical studies across numerous populations (e.g., [6]) have shown that users with low self-efficacy tend to shy away from unfamiliar features designed to help them. Table IV identifies the two males and two females whose self-efficacy scores from the background questionnaire were below the medians for their gender (marked as "Self-Eff"). The Idea Garden's suggestion appeared to interact with these low self-efficacy participants in both negative and positive ways. For example, M3, whose self-efficacy score was among the lowest in the male participants, when the feature came up, acknowledged the suggestion as "a direction to repeat" but did not follow through with it, perhaps due to his low self-efficacy: "*This is hard. I don't know what I'm doing*". On the other hand, the participant with the lowest self-efficacy score, F5, was able to make use of the suggestion's script snippet and eventually edit the snippet to succeed

beyond the second subtask accomplishing the third subtask as well. Thus, in M3's case, the suggestion did not allay his concerns about his abilities, and may even have exacerbated them, whereas in F5's case, the suggestions may have reassured her that she was moving in the right direction.

C. RQ3: From Barriers To Learning

To assess possible learning, we considered evidence of learning to be the temporal sequence <barrier, Idea Garden use, correct answer>. That is, a participant first had to show a lack of knowledge by running up against a barrier, then had to turn to the Idea Garden for help, and finally had to answer post-test questions correctly. Our rationale was that those who ran into barriers were, by definition, demonstrating a knowledge gap in at least one of the concepts, patterns, and/or strategies.

Table V and Table VI summarize the results for the knowledge items relevant to the Composition barrier and the More-Than-Once barrier, respectively. The strategy questions included both broad, open-ended questions that tested whether the participant would think to apply the strategy in planning how to proceed with a new task, and more specific questions that tested whether the participant could apply the strategy either in the use of specific Idea Garden features or when facing a specific situation within the new task. The feature-specific questions for analogy were presented to only the participants who saw the corresponding feature during the task, and thus, those questions are treated separately in the tables.

Overall, every participant who encountered a barrier and then used the Idea Garden demonstrated understanding of all the relevant programming concepts (Dataflow and Iteration) and one of the patterns (Webpage-as-Component) during the interview; however, only those who subsequently overcame barriers *on their own* (i.e., without hints from the researcher) demonstrated understanding of the Repeat-Copy-Paste pattern, the ability to solve the more open-ended Analogy and Generalization problems, and the ability to apply the Generalization strategy in a specific situation within a task.

VI. Discussion and Conclusion

Our results showed that 9 of the 10 participants who encountered barriers and used the Idea Garden, subsequently overcome at least one of their barriers on their own. Moreover, in the post-test, those 9 demonstrated understanding of all the programming concepts, patterns, *and* strategies relevant to the barriers they overcame on their own. Their difficulties with the barriers prior to using the Idea Garden, combined with their subsequent success after using the Idea Garden, and their demonstrated understanding during the post-test, triangulate to suggest they indeed learned from their experience.

Minimalist Learning Theory [10], which inspired our design of the Idea Garden (Section III.B), provides a basis for understanding its features' effectiveness. This theory argues that active users will be more likely to use and profit from learning-related resources that are *situated* within their task, rather than in some other program or resource, such as a tool or tutorial. Recall that few participants referred back to the tutorial examples; we hypothesize that this is because the examples were external to the task at hand.

Another way to understand these results is in terms of Attention Investment (a model of cost, benefit, and risk), which posits that users will be more inclined to learn new abstractions if they perceive that doing so requires low up-front costs and provides substantial benefits [3]. CoScripter has numerous such abstractions, including tables and repetition, that are needed for mashup tasks. In the case of the Idea Garden, the cost to venture forward starts with reading a tool tip—a cost that most users perceive to be low. The Idea Garden's suggestions are concrete enough to keep users' further estimates of cost aligned with actual costs. The potential benefit is that overcoming the barrier will allow the user to move ahead with their task. In contrast, external tutorials or examples require switching to other documents and sifting through material to find the

Table V. Interview results for questions related to the Composition barrier. ✔: answered a question correctly. -: showed no evidence of the knowledge item. ✘: incorrect use of the knowledge item. ?: did not answer. NA: participant was not asked that question. Shaded: received help from researcher relating to the knowledge item.

Knowledge Item		Overcame barrier on their own			Did not overcome barrier on their own			Demonstrated Knowledge
		F1	F4	M3	F3	M1	M2	
Dataflow Concept (problem required it)		✔✔✔✔✔	✔✔✔✔✔	✔✔✔✔✔	✔✔✔✔✔	✔✔✔✔?	✔✔✔✔✔	6/6
Webpage-as-Component Pattern (problem required it)		✔	✔	✔	✔	✔	✔	6/6
Analogy Strategy	(context: broad, open-ended task planning)	-	✔	✔	-	-	-	2/6
	(context: using Table Tooltip)	✔✔	-✘	NA	✔✘	NA	-✘	2/4
	(context: using Second Webpage)	NA	NA	✔✔✘	✔✔?	✔✔✘	✔✔✔	4/4

Table VI. Interview results for questions related to the More-Than-Once barrier (Symbols from Table V).

Knowledge Item		Overcame barrier on their own							Did not overcome barrier on their own		Demonstrated Knowledge
		F1	F2	F3	M2	F5	M4	F6	M1	M3	
Iteration Concept (prediction/comprehension questions)		✔✔✔✔	✔✔✔✔	✘?✔✔	✔✔✔✔	✔✔✔✔	✔✔✘✔	✔✔✘✔	✔✔✔✔	✔✘✔✔	9/9
Repeat-Copy-Paste Pattern (problem required it)		✔	✔	✔[1]	✔	✔	✔	✔	✘	✘	7/9
Generalization Strategy	(context: broad, open-ended task planning)	✔-	- -	- ✔	- -	- -	- -	- -	- -	- -	2/9
	(context: specific situation within a task)	✔	✔	✔	✔	✔	✔	✔	✘	✘	7/9

[1] Researcher helped with first copy/paste, then participant invoked Idea Garden to overcome the rest.

relevant parts, with a potential risk that relevance to the task at hand will still not be clear.

Recall also that, by design, the Idea Garden provides relevant but *incomplete* suggestions. Our hope was that this incompleteness would engage users intellectually and encourage them to apply strategies such as Analogy that would stay with them. Their success with overcoming barriers, combined with their correct answers to the analogy questions in our post-test, suggest that this approach showed merit.

Nonetheless, a few participants struggled with Analogy, which prevented them from overcoming some barriers. This limitation suggests that analogies need to be more easily recognizable and applicable to the task at hand. One approach might be to present users unable to overcome a barrier with more analogies. For example, the Idea Garden might focus a user who struggles with a barrier on analogous web pages (as in the Table Tooltip feature) or analogous relationships (such as input-output relationships) to help the user reframe their concept of a webpage's capabilities.

Although the Idea Garden's context-sensitive tooltips were designed to make suggestions more helpful to users, over-contextualizing assistance to the task at hand runs the risk of limiting transferability of learning to other situations. For many participants, it appears that this generally was not a problem with the Idea Garden, as these participants were able to successfully answer post-test questions that asked them to describe how they would perform another programming task. It is an open question regarding the extent to which this apparent learning will lead to quantitative improvements in users' ability to correctly complete other programming tasks.

Overall, our results suggest that, by being available to support learning, the Idea Garden encouraged users actively trying to accomplish a programming task to learn along the way. While tools that automate away some barriers may help in the short term, this help may come at the expense of skills users will need to handle similar difficulties that may arise later. Our study's results suggest that the Idea Garden approach may help users to overcome their barriers now *and* in their future.

ACKNOWLEDGEMENT

We thank Forrest Bice and Hannah Adams for their assistance with implementing the Idea Garden prototype. This work was supported in part by NSF grant 0917366.

REFERENCES

[1] L. Anderson, D. Krathwohl, P. Airasian, K. Cruikshank, R. Mayer, P. Pintrich, J. Raths, and M. Wittrock, *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*, Abridged Edition, 2nd ed. Allyn & Bacon, 2000.

[2] A. Bandura, Self-efficacy: Toward a unifying theory of behavioral change, *Psychological Review* 8(2), 191–215, 1977.

[3] A. Blackwell, First steps in programming: A rationale for attention investment models, *IEEE HCC*, 2– 10, 2002.

[4] J. Brandt, P. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer, Two studies of opportunistic programming: Interleaving web foraging, learning, and writing code, *ACM CHI*, 1589–1598, 2009.

[5] J. Bransford, A. Brown, R. Cocking. *How People Learn: Brain, Mind, Experience, and School*, Expanded ed., National Academy Press, 2000.

[6] M. Burnett, S. Fleming, S. Iqbal, G. Venolia, V. Rajaram, U. Farooq, V. Grigoreanu, M. Czerwinski, Gender differences and programming environments: Across programming populations, *ACM-IEEE ESEM*, 2010.

[7] J. Cao, Y. Riche, S. Wiedenbeck, M. Burnett, and V. Grigoreanu, End-user mashup programming: Through the design lens, *ACM CHI*, 1009-1018, 2010.

[8] J. Cao, K. Rector, T. Park, S. Fleming, M. Burnett, and S. Wiedenbeck, A debugging perspective on end-user mashup programming, *IEEE VL/HCC*, 149–156, 2010.

[9] J. Cao, S. Fleming, and M. Burnett, An exploration of design opportunities for 'gardening' end-user programmers' ideas, *IEEE VL/HCC*, 35-42, 2011.

[10] J. Carroll. *Minimalism Beyond the Nurnberg Funnel*. MIT Press, 1998.

[11] J. Carroll ,M. Rosson, Paradox of the active user, *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*, MIT Press, 1987.

[12] C. Chambers and C. Scaffidi, Struggling to excel: A field study of challenges faced by spreadsheet users, *IEEE VL/HCC*, 187–194, 2010.

[13] D. Compeau and C. Higgins, Computer self-efficacy: Development of a measure and initial test, *MIS Quarterly* 19(2), 189–211, May 1995.

[14] A. Cypher, M. Dontcheva, T. Lau, and J. Nichols. *No Code Required: Giving Users Tools to Transform the Web*. Morgan Kaufmann, 2010.

[15] P. Diaz, M. Rosson, I. Aedo, and J. Carroll, Web design patterns: Investigating user goals and browsing strategies, *IS EUD*, 186–204, 2009.

[16] R. Ennals, E. Brewer, M. Garofalakis, M. Shadle, and P. Gandhi, Intel Mash Maker: Join the web, *SIGMOD Rec.* 36(4), 27–33, Dec. 2007.

[17] T. Green and M. Petre, Usability analysis of visual programming environments: A 'cognitive dimensions' framework, *J. Visual Langs. Computing* 7(2), Jun. 1996.

[18] P. Gross and C. Kelleher, Non-programmers identifying functionality in unfamiliar code: strategies and barriers, *J. Visual Langs. Computing* 21(5), 263–276, Dec. 2010.

[19] M. Guzdial, Education: Paving the way for computational thinking, *Commun. ACM* 51(8), 25–27, Aug. 2008.

[20] B. Hartmann, D. MacDougall, J. Brandt, and S. Klemmer, What would other programmers do: Suggesting solutions to error messages, *ACM CHI*, 1019–1028, 2010.

[21] C. Hundhausen and J. Brown, What you see is what you code: A 'live' algorithm development and visualization environment for novice learners, *J. Visual Langs. Computing* 18(1), 22–47, Feb. 2007.

[22] A. Ioannidou, A. Repenning, and D. Webb, Using scalable game design to promote 3D fluency: Assessing the AgentCubes incremental 3D end-user development framework, *IEEE VL/HCC*, 47–54, 2008.

[23] C. Kissinger, M. Burnett, S. Stumpf, N. Subrahmaniyan, L. Beckwith, S. Yang, and M. B. Rosson, Supporting end-user debugging: What do users want to know? *AVI*, ACM, 135–142, 2006.

[24] A. Ko, B. Myers, and H. Aung, Six learning barriers in end-user programming systems, *IEEE VL/HCC*, 199–206, 2004.

[25] S. Kuttal, A. Sarma, G. Rothermel, History repeats itself more easily when you log it: Versioning for mashups, *IEEE VL/HCC*, 69–72, 2011.

[26] J. Lin, J. Wong, J. Nichols, A. Cypher, and T. Lau, End-user programming of mashups with Vegemite, *ACM IUI*, 97–106, 2009.

[27] R. Miller, M. Bolin, L. Chilton, G. Little, M. Webber, C.-H. Yu, Rewriting the web with Chickenfoot, In *No Code Required: Giving Users Tools to Transform the Web*, Morgan Kaufmann, 2010.

[28] G. Polya. *How to Solve It: A New Aspect of Mathematical Method* 2nd ed., Princeton University Press, 1971.

[29] M. Resnick, Sowing the seeds for a more creative society, *Learning and Leading with Technology* 35(4), 18–22, 2007.

[30] D. Schön. *The Reflective Practitioner: How Professionals Think in Action*, Basic Books, 1983.

[31] C. Seaman. Qualitative Methods in Empirical Studies of Software Engineering. *IEEE Trans. Softw. Eng.* 25(4), 557–572, 1999.

[32] J. Wong and J. Hong, Making mashups with Marmite: Towards end-user programming for the web, *ACM CHI*, 1435–1444, 2007.

[33] N. Zang and M. Rosson, Playing with information: How end users think about and integrate dynamic data, *IEEE VL/HCC*, 85–92, 2009.