Lost while Searching: Difficulties in Information Seeking among End-User Programmers

Brian Dorn
Department of Computer Science
University of Nebraska at Omaha
6001 Dodge Street
Omaha, NE 68182
bdorn@unomaha.edu

Adam Stankiewicz and Chris Roggi
University of Hartford
200 Bloomfield Avenue
West Hartford, CT 06117
astankiew@hartford.edu,
chris.roggi@gmail.com

ABSTRACT

End-user programmers, those who write code but lack formal training in computer science, are often reliant on various tools such as API documentation or searching the Web for information in order to complete a specific task. This study examines the information foraging behaviors of a group of web and graphic designers engaged in a series of code modification tasks. We find that users were largely unsuccessful in their foraging activities, with few information seeking events resulting in noticeable changes to participants' source code. Participants viewed remarkably few results generated by their queries and rarely refined queries multiple times. However, these kinds of activities were positively correlated with task success metrics. We conclude with a discussion of the study's results and their implications on the design of future programming environments and search tools for end-user programmers.

Author Keywords

end-user programming, information foraging, search behaviors, novice programmers

INTRODUCTION

Programming and debugging can be daunting tasks, especially when one has minimal experience with the language syntax and one's knowledge of the underlying principles is incomplete. Unfortunately these are precisely the circumstances in which the vast majority of people who write code find themselves (Scaffidi, Shaw, & Myers, 2005). In particular, we are interested in communities of *end-user programmers*. By definition such programmers lack formal training in computer science, but nonetheless take up scripting and programming as a means to increase efficiency or effectiveness in the context of their work (Nardi, 1993; Beringer, 2004). Canonical examples of end-user programmers include accountants who write spreadsheet formulas and macros (Burnett et al., 2003), biologists who create programs to simulate models or facilitate data analysis (Segal, 2007), and graphic designers who produce scripts to automate repetitive and tedious tasks (Dorn & Guzdial, 2006). Disciplinary differences aside, a common characteristic of people in these examples is that their formal academic training is in something other than computing. Their knowledge of the computing fundamentals, and more specifically programming, is built largely from self-taught experiences and advice from peers.

ASIST 2013, November 1-6, 2013, Montreal, Quebec, Canada.

Ko, Myers, and Aung (2004) cataloged several learning barriers encountered by end-user programmers. These six classes of problems are briefly summarized below:

- **design:** inherent difficulties in conceptualizing a solution to the problem
- **selection:** issues locating and selecting the relevant programming components and interfaces from those made available by the system
- **coordination:** problems properly composing selected components into a workable solution
- **use:** properties of the programming interface that obscure an element's usage or its effect
- **understanding:** difficulties that arise as a result of a mismatch between a program's external behavior (e.g., at runtime) and learner expectations
- **information:** challenges caused by an inability to inspect a program's internal behavior to test learner hypotheses

Researchers have long sought to address many of these barriers through the design of new programming languages and environments (e.g., Burnett et al., 2001; Wilson et al., 2003; Pane, Myers, & Miller, 2002; Leshed, Haber, Matthews, & Lau, 2008; Ko et al., 2011). Such approaches seek to change the nature of the programming task to eliminate or reduce these barriers. However, one might also conceptualize these barriers as fundamental to the task of learning to program resulting from information and knowledge deficits. Indeed, many of them had been identified earlier in computing education research on novice programmers (Spohrer & Soloway, 1985; Du Boulay, 1989; Green & Payne, 1984).

Viewed in this light, it becomes important to understand how end-user programmers try to overcome these knowledge gaps by using the common resources at their fingertips like application programmer interface (API) documentation and the Web. By identifying ways to support their existing informa-

tion foraging behaviors, we may also discover ways to enhance the effectiveness of their self-teaching strategies.

In this paper we investigate the information seeking behaviors of a group of 19 web and graphic designers engaged in programming and debugging tasks within the Photoshop Extendscript Toolkit. Participants worked through a series of code-oriented tasks related to a pre-existing project in a usability laboratory. They were given various digital resources along with unrestricted access to the Web. Using screen capture and logfile data, we specifically address the following two research questions:

RQ1 What foraging and search behaviors are employed by end-user programmers to overcome learning barriers?

RQ2 To what extent do information seeking behaviors correlate with various measures of success in completing scripting tasks?

The remainder of this paper proceeds as follows. First we provide an overview of related work in information seeking behaviors, particularly those tied to answering questions about source code. Next, we describe methodological details about our study design and data analysis approach. We then present the results of the study and conclude with a discussion of our findings and their implications for the design of new scaffolds to future systems supporting the information seeking tasks of end-user programmers.

RELATED WORK

Tasks and Information Seeking

A large body of prior work examines information retrieval behaviors, particularly those related to users' search strategies and their relationship to the task at hand. In a study examining Web information finding on a university website, Gwizdka and Spence (2006) noted a positive correlation between task difficulty and both the number of pages visited and time spent per page. Aula, Khan, and Guan (2010) used a series of closed-ended information tasks to examine the search behaviors of 179 participants. On tasks identified as difficult in this study, users exhibited a wider range of queries, incorporated more Boolean search operators, and spent more time examining search results (both in terms of absolute time and proportionally to the total time for each task). J. Liu, Gwizdka, Liu, and Belkin (2010) examined the interplay between task difficulty and task type for typical Web information tasks. Their work studied user behaviors in single-fact finding tasks, multiple-fact finding tasks, and multiple-piece information gathering tasks. They found that participants expended more effort on the difficult tasks generally and that participants' behaviors for where search effort expended (ie., examining results pages vs examining content pages) was different for the different types of tasks.

Other research has focused on the relationships between search behaviors and participants' underlying domain knowledge. Such efforts build on work in cognitive psychology and the learning sciences that distinguishes differences between novice and expert behaviors in problem solving tasks (for an in-depth review of these issues, see (Bransford, Brown, & Cocking, 2000)). Novices (ie., those with low levels of domain knowledge) tend to view knowledge as a collection of isolated facts rather than an integrated framework of concepts, and novices are more likely to attend to surface-level features of problems rather than fundamental principles when seeking solutions (Bransford et al., 2000). Studies in a variety of different domains ranging from medicine (Wildemuth, 2004) to computer science (White, Dumais, & Teevan, 2009) to football (Duggan & Payne, 2010) have shown significant differences in the search behaviors of novices and experts. Wildemuth (2004) found that medical experts were more efficient in information foraging tasks and exhibited more sophisticated query refinement approaches. Duggan and Payne (2010) found a positive correlation between expertise and search accuracy. In a study of four different disciplines, White et al. (2009) noted strong relationships between search behaviors and expertise, but also that the relationships did not hold constant across the disciplines. Most central to our work, greater levels of computer science expertise were tied to shorter page display times—those with more CS knowledge seemed better equipped to extract the needed knowledge from the result page more quickly (White et al., 2009).

C. Liu, Liu, Cole, Belkin, and Zhang (2012) investigated interaction effects between task difficulty and expertise as it pertains to information seeking. Their study asked undergraduate and graduate students from a variety of the health sciences disciplines to complete tasks using a custom medical information website. They found that increased task difficulty related to increased dwell time on search results pages and increased page views. They found significant interaction effects between expertise and task difficulty and concluded that novices and experts indeed exhibit unique foraging strategies. They hypothesize that this could be due to experts' better ability to evaluate search results for their relevance to the task at hand while novices have inefficient search strategies that rely on pattern-matching between search results and surface-level details of the task descriptions.

This prior work is particularly salient for our study because of the unique situation in which end-user programmers find themselves. By definition, they lack extensive domain knowledge in programming languages and computer science concepts (Nardi, 1993; Beringer, 2004). However, the tasks that they are attempting to accomplish often involve difficult, complex code operations (see, e.g., Dorn & Guzdial, 2006). Our work builds on this prior body of knowledge to explore the particular behaviors exhibited by this group of users in order to identify challenges and potential avenues to better support their information needs. We also extend this line of research by using task performance as another dimension along which to characterize information gathering behaviors.

Foraging for Answers about Code

Another strand of related work involves exploration of how programmers understand existing code and seek information (either within the code itself or using third party reference sources like the Web) to aid in development tasks. Much of the information foraging research applied to programming tasks has investigated the behaviors of professional develop-

ers or those who have significant coursework in computer science. Ko, Myers, Coblenz, and Aung (2006) explored the behaviors of developers engaged in a series of software maintenance tasks on unfamiliar code, a typical activity for professionals. They identified common behaviors including searching the code base (both manually and using search tools), relating code to other relevant code sections by following function dependencies, and collecting helpful pointers to potentially useful code sections within the programming environment. Similarly, LaToza, Garlan, Herbsleb, and Myers (2007) cast program comprehension as an exercise in fact finding. Their study involved participants who all had prior employment in the software industry (defined as two internships or better), and grouped them either as novices or experts based on the degree of their experience. They found that experts were better able to make changes to the source code that addressed root causes of problems and that experts were better at identifying relevant portions of the source code. Novices, on the other hand, focused on changes that attempted to remedy the visible symptoms of the problems and were more likely to describe program behavior in a line-by-line fashion.

While directly concerned with information seeking in source code, the findings of these studies are consistent with both the aforementioned findings about domain expertise and differences between novices and experts engaged in knowledge tasks. Further, in a code comprehension study using the Storytelling Alice environment, Gross and Kelleher (2010a) found that non-programmers exhibit similar information foraging behaviors to those noted by Ko et al. (2006). Because the end-user programmers of interest in our work must regularly make sense of unfamiliar code, we expect their behaviors to in many ways mirror that of the novices in the work of LaToza et al. (2007). However, because it is likely that such end-user programmers have even less knowledge about the syntax and semantics of basic programming constructs, we are concerned with how they use resources beyond the source code to answer questions and solve problems.

Typical debugging and software development tasks involve a range of information gathering activities, and Brandt, Guo, Lewenstein, Dontcheva, and Klemmer (2009) note that in opportunistic programming domains (where the majority of code is done with little pre-planning) code writing is interleaved with Web searches and learning "on-the-fly." They characterized a number of search behaviors for developers by analyzing queries within the Adobe developers' portal (Brandt et al., 2009). More recent work by Bajracharya and Lopes (2012) examined the search patterns within Koders, an Internet search engine that explicitly indexes source code. They found that 93% of queries made no use of advanced Boolean search operators, 55% of search sessions contained only one query, 57% of queries consisted exclusively of code, 33% made use of only natural language terms, and 9% contained a mixture of the two. Their findings are generally similar to those identified by Brandt et al. (2009), with the exception of a higher occurrence of exclusively code queries likely due to the code-only nature of the Koders system. While these studies both provide illustrative details about foraging for details about code on the Web, end-user programmers may be less likely to be aware of the more targeted resources available to answer code related questions. In our work, we examine information seeking in a less restrictive fashion in order to explore what web-based resources these users explore and how they use them to answer questions.

METHOD

The data of interest in this paper was generated during a 2-hour study session in a controlled usability lab. In each individual session, participants were asked to complete a series of tasks in an existing Photoshop scripting project written in ExtendScript (Adobe's variant of JavaScript). These tasks consisted of code comprehension, bug fixing, and feature extension. The sessions were designed to establish participants' performance on the tasks using only their pre-existing knowledge and current information seeking strategies on the Web. This data also provided a baseline for a comparison study to evaluate the effectiveness of a newly developed case-based learning aid (see (Dorn, 2011) for more details).

Artifacts generated by the participants during testing sessions were used to assess their code quality and their conceptual understanding of underlying computing concepts. Participants' actions on the computer were automatically recorded using the Morae usability testing suite, and the resulting log files provided a detailed history of their interactions with multiple applications.

Recruitment

Participants were primarily recruited from emails to local interest groups related to professional Web/graphic design and classified advertisements. A minority of participants were also recruited from undergraduate student (non-Computer Science) groups. Volunteers were screened via email to confirm that they met study criteria. Those who had no prior experience with scripting or programming were declined, as a basic ability to read and understand code was required for participation. Further, volunteers with no clear connection to digital media, or Web/graphic design either by profession or coursework were excluded. These procedures ensured that our pool of participants was familiar with scripting, involved in the field of Web/graphic design, but not professionally trained in software development. All participants were compensated with a $75.00 Amazon gift card.

Participant Demographics

At the outset of the session, participants were asked to complete a demographic survey about their scripting expertise and background. In total, nineteen participants completed the study described here. Of these, 11 were men and 8 were women, and participants represented a wide cross-section of the recruitment pool. About a third of participants (6/19) indicated a primary occupation in the Web or graphic design industries, about 42% (8/19) were full-time post-secondary students, and the remaining 5 participants reported a combination of the two (e.g., a part-time student with an industry position). A variety of job titles were given, but careers in photography, Web development, graphic design, or other design disciplines accounted for approximately 68% (13/19) of the total.

Participants had a variety of prior experience with image editing and scripting tools. Experience with Photoshop ranged from 1 to 18 years with an average of 6.0 years ($\sigma = 4.7$), with participants reporting that they used it 9.2 hours per week on average ($\sigma = 14.6$). Participants had less exposure to scripting or programming with an average of 4.5 years ($\sigma = 4.0$) of prior experience. On a scale from one (novice) to five (expert), the average self-reported rating of scripting expertise was 2.5 ($\sigma = 1.0$). Thus, on the whole, participants were experienced Photoshop users with at least basic knowledge of programming.

On the demographic survey, participants indicated a high user preference for resources like tutorials, online documentation, and code examples, similar to what has been observed elsewhere (Dorn & Guzdial, 2006, 2010; Rosson, Ballin, & Rode, 2005). Every participant said they would be likely or very likely to consult tutorials or frequently asked question documents when attempting to learn something new, and all but one said they would refer to existing examples from which they could borrow ideas or code. Less than half (8/19) said they would attend a class or seminar on the subject and only two would be likely to call technical support phone numbers.

Study Sessions and Tasks

Following the administration of the demographic survey described earlier, participants were introduced to the specific project they would be working on in this task and shown the programming environment provided with Photoshop. The project involved the automatic extraction of meta-data from a collection of photos to produce a comma-delimited file with the data. Participants were given time to review the provided materials and then were asked to complete two warm-up tasks to familiarize themselves with the project and programming environment. Research personnel assisted as necessary during the warmup tasks to build participants' confidence. After completing warm-up tasks, participants were given 90 minutes to complete 6 individual tasks using only information found on the Web or in the documentation materials accessible through the IDE's help menu.

The six assigned tasks were designed to require the use a variety of different introductory computing concepts including selection, indefinite loops, functional decomposition, exception handling, importing external code libraries, and recursion. Each task targeted exactly one unique concept, with the exception of one task involving both functional decomposition and importing code modules.

The tasks were divided into two or more related sub-tasks designed to assess the intended construct from different levels of sophistication along Bloom's Taxonomy (Anderson et al., 2001) of knowledge. Subtasks were classified as "describe," "strategize," or "code" questions based on the nature of the prompt. Describe subtasks asked participants to run the script in a particular manner and describe what happened. Strategize subtasks asked participants to devise a strategy for preventing the error that occurred and outline the programming technique they would use in plain English. Lastly, code subtasks asked participants to implement their given strategy within the project. The three question types made it possible to investigate both conceptual knowledge about a topic in addition to practical code development ability. This is important because it was foreseeable that a participant could correctly understand what was needed to complete a task, but struggle with elements of JavaScript syntax in implementing their idea.

Data Coding Processes

Several different techniques were used to construct the data sets for our subsequent analysis. Data regarding information seeking behaviors was extracted through careful examination and coding of participant screen capture videos. In order to assess the success of participants in completing the assigned tasks, additional coding procedures were applied to the handwritten responses to the strategize subtasks and the source code written by participants for code subtasks. We discuss each of these in turn.

Screen Capture and Log Data

Information seeking behaviors were identified by examining the 31.3 hours of screen capture data collected during the study. Two of the authors devised a coding rubric to identify and categorize events related to information foraging in relationship to the assigned tasks. The rubric was applied to the session data from one of the 19 participants (accounting for approximately 5% of the total data set) by two independent raters. The raters then reviewed the resulting codes, discussed discrepancies, and revised the rubric for clarity. They then re-rated the original subset of video data along with that of one more participant and collaboratively reviewed the results ($\approx 10\%$ of the data). This iterative process continued until the rubric was sufficiently clear to avoid misapplication. At that time, one person coded the rest of the data set.

Table 1 outlines the coding rubric used to classify foraging events on multiple dimensions. A discrete event was said to begin when a participant left the programming environment and opened a web browser or another digital information resource. In many situations a single event only captures part of a larger information seeking sequence, as in the case where a participant performs a web search followed by series of follow-up searches to narrow down the results. Here, each search query and possible viewing of results would be treated as an individual event. Therefore, we also aggregated such series of events (determined by whether an event was considered a refinement, as defined in Table 1) into information seeking "sessions" to provide a higher-order depiction of foraging patterns. In our presentation of the results, we will use the term *event* to refer to the finer-grained data and *session* to refer to the aggregated data.

Assessing Participant Code Quality

There were four "code" subtasks completed by each participant. To analyze the quality of the code produced for these subtasks we developed a rubric with four ordered categories to be applied to each of the final scripts. The categories, from most correct to least, were: (4) Code functions correctly; uses the intended construct; closely resembles the ideal solution. (3) Code functions correctly; uses the intended construct; but also includes unnecessary additional constructs that could be removed or otherwise simplified. (2) Code does not function

Dimension	Code Levels	Description
Behavior type	search, browse	Search is an attempt to find information using keywords or phrases inserted into the text-area of a search tool (either internal to the application software or within a website). Browse involves use of a built-in selection feature (a list, collapsible menu, or hyperlink) within possibly useful content. Each browse event must yield a result displayed in the primary content area of the tool.
Resource used	IDE, API, Web, PDF	This indicates the application in which the event took place. IDE indicates a event within the ExtendScript coding interface (e.g., using the built-in find function to search over the source code); API indicates an event within the interactive language reference known as the object-model viewer; Web is an event occurring within a web browser; and PDF refers to an event occurring within one of the reference PDF documents provided by Adobe.
Query type	code, natural language, hybrid	Examines the search query string contents. If it contains exclusively program syntax, then code (e.g. "`File.openDialog`"). If it contains no code, then natural language (e.g., "javascript delete confirmation dialog"). If it uses both code and English words or phrases, then hybrid (e.g., "`application.open` fail javascript").
Refinement event	yes, no	Yes, if event is temporally close to a prior event (within 1 minute) and query strings or browse paths contain some shared substring. For example, if an initial query of "cs4 extendkit functions" is followed by a query of "cs4 extended kit functions" then second query event is considered a refinement.
Successful event	yes, no	Yes, if upon completion of a seeking event, participant made changes pertaining to information viewed and the part of the problem that the participant was experiencing no longer exists. Examples include copy/pasting code, adding new syntax related to results viewed, or a verbal utterance indicating participant found what s/he was searching for.
Results viewed	number	A count of the total number of results/pages viewed during this event, not including the search results overview page.

Table 1. Information Seeking Event Coding Rubric

correctly but uses the intended construct. (1) Code does not function correctly and does not use the intended construct. (0) No code edited or inserted.

Two independent raters applied the rubric to all 76 script excerpts produced (4 excerpts per participant). Ratings were compared and any disagreements were collaboratively reconciled between the raters to produce a final value.

Assessing Conceptual Response Quality

We also sought to assess participants' ability to answer open-ended, conceptual questions about programming concepts. Handwritten responses to the five "strategize" subtasks serve as our indicators of conceptual understanding. These prompts asked participants to outline a programming technique or concept that would be applicable in the current situation and would allow them to overcome an issue in the code.

Similar to the assessment of code performance, we developed a rubric for judging the quality and correctness of conceptual responses. Again, two independent raters applied the rubric to all 95 responses (five responses per participant) and collaboratively reconciled any disagreements. The final rubric used for each question had the following four categories, ordered from most to least correct: (3) Response correctly identifies the intended construct by name, using normative terminology. (2) Response correctly describes an approach that makes use of the intended construct, but does not explicitly use normative terminology in the answer. (1) Response does not address the intended construct, but does exhibit an algorithmic or programmatic approach to solving the problem that could work under limited conditions. (0) No answer given, or response was nonsensical or unrelated to the issue being addressed.

RESULTS

In this section we present results organized by our two overriding research questions. First, we examine the data to characterize general information seeking behaviors exhibited by our participants. Then we turn to a statistical analysis of the relationship between these behaviors and underlying success metrics for the assigned tasks.

Information Seeking Behaviors

Table 2 contains basic statistics for the observed information seeking behaviors including both search and browse activities. As can be seen in the table, participants engaged in a large number of total seeking events, but viewed relatively few of the results that were presented following their search queries or interactive browsing. In fact, no result was viewed at all in response to 53% of the search events.

	count	avg num results viewed	num refinements	successes
Total events	539	0.81 ($\sigma = 0.95$)	239 (44.3%)	28 (5.2%)
Search events	366 (67.9%)	0.73 ($\sigma = 1.14$)	176 (48.1%)	24 (6.6%)
Browse events	173 (32.1%)	0.98 ($\sigma = 0.13$)	63 (36.4%)	4 (2.3%)
	count	**avg num results viewed**	**avg num refinements**	**successes**
Search sessions	190	1.41 ($\sigma = 1.87$)	0.93 ($\sigma = 1.59$)	24 (12.6%)
Browse sessions	110	1.55 ($\sigma = 1.23$)	0.57 ($\sigma = 1.23$)	4 (3.6%)

Table 2. Information Seeking Events and Sessions

Participants exhibited only low levels of persistence when it came to follow-up information seeking behaviors, like query refinement. Less than half of the events generated were refinements in response to prior events, and the average number of refinements per session was 0.93 for search activities and 0.57 for browse activities. While the longest search session contained 9 subsequent query refinements, 57.9% of search sessions ended with the first query alone. Browse sessions

contained even fewer follow-ups, with 81% ending after the first event.

Taken together these results suggest that our participants had considerable difficulty in formulating productive queries and may have become discouraged quickly in the face of poor search results. Assessing the relevance of search results is a complex skill (Teevan, 2008), and it is made more difficult for end-user programmers whose knowledge of the task at hand is only partially formed (Ko et al., 2004).

We then examined participants' predilections for engaging in search or browse behaviors by computing the relative percentage of each given all of the information seeking events they generated. Figure 1 depicts the results. Generally speaking, the majority of participants favored text-based search activities to interactively browsing information resources. Only three engaged in browsing activities more than searching activities. On average a little more than two-thirds of all information seeking utilized a search engine, either internally to the API software or externally on the Web.

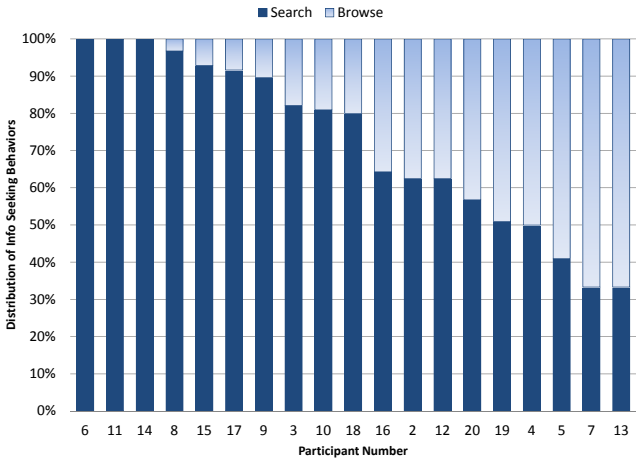

Figure 1. Search vs Browse Split by Participant

Given the varying difficulty levels of the tasks in the study and the conceptual material covered, we also investigated to what degree search versus browse behaviors remained consistent across the tasks. As shown in Figure 2 the 2:1 search-to-browse ratio generally holds across the six tasks, with the final task being a notable outlier. However, task six only resulted in 3 total information seeking events which was considerably (by an order of magnitude) fewer than any of the other tasks. Ironically, this task proved the most difficult for participants as it involved devising a recursive solution to an existing bug in the project. Recursion is well-known as a challenging concept for novices (Wiedenbeck, 1988) and has been identified as one of the least well understood by web and graphic design end-user programmers (Dorn & Guzdial, 2010). It may have been the case that our participants were simply unaware of their knowledge gap and their need to learn more, which is consistent with prior research showing novices often don't know what they don't know (Bransford et al., 2000).

In addition, we examined differences in behaviors related to the resources used. The distributions of search and browse

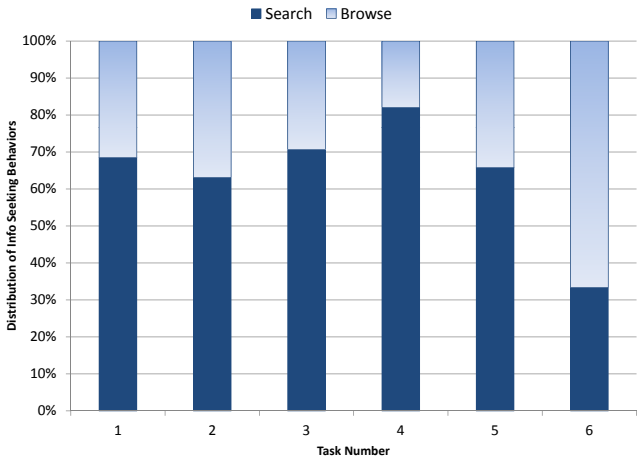

Figure 2. Search vs Browse Split by Task

events across different resources is given in Table 3. We noted a statistically significant difference in the distribution of events across the various tools (Fisher's exact test, $p = 0.008$). Very few information seeking events occurred within the development environment or external PDF documentation for the scripting tool. Search events were split between the interactive API tool (known as the Object-Model Viewer) and Web-based searches, with the Web garnering slightly more use. Only two Web-based search events utilized internal, site-specific search engines, and the overwhelming majority of the rest used Google's general purpose search engine. Browse events were 10 times more likely in the API tool than by navigating Web-based resources using hyperlinks. The Object-Model Viewer was structured such that clicking to drill-down through categorical information (e.g., packages, object types, methods) was quite natural, and we observed many participants having difficulty with its internal search engine.

	IDE	PDF	API	Web
Search events	1.6%	1.9%	43.2%	53.3%
Browse events	0.0%	1.2%	90.2%	8.7%

Table 3. Information Resource Usage

Lastly, we also noticed differences in behaviors related to the specific query types made. Search query strings were categorized based on whether their contents included JavaScript code or not. We found that 38.3% of all search queries were strictly code, 42.6% contained no source code (i.e., they were natural language searches), and 19.1% contained a mixture of the two. When events were aggregated into sessions, we found the average search session included 0.74 code queries, 0.82 natural language queries, and 0.37 hybrid queries. While we did not explicitly examine query contents relative to the participants' intent, many of the code query contents were copy/pasted directly from the project's source code.[1] This resulted in difficulties for participants because a search involving local variable and function names was unlikely to yield useful results, particularly when searching the Web with Google (as was most common).

[1] See (Dorn, 2010) for a full catalog of query contents.

Relationship to Success Metrics

In this subsection, we explore relationships between the observed information seeking behaviors and participant performance on the assigned tasks. The code and handwritten responses generated by participants were scored using the rubrics described earlier. Participants had difficulties with the various tasks in general as exhibited by the low averages seen in Table 4, but individual scores from each of the ranges (0-4 for code, 0-3 for concept) were present. Some tasks did not include a code writing sub-task or a strategize sub-task; these are indicated as not applicable in the table, and in the results to follow we only consider data from the tasks for which there are scores in the analysis of the relationship between information seeking behaviors and code/concept score. On tasks where data for both are available, we observed a statistically significant strong positive correlation between code and concept scores (Spearman's $\rho(57) = 0.65, p < 0.01$). In other words, participants demonstrating higher degrees of conceptual knowledge were able to enact that knowledge to produce more correct code solutions, as might be expected.

	T1	T2	T3	T4	T5	T6
mean code score; max of 4 (stdev)	2.26 (1.37)	1.95 (1.31)	1.63 (1.64)	n/a	1.68 (1.34)	n/a
mean concept score; max of 3 (stdev)	n/a	1.47 (0.90)	1.53 (0.84)	1.47 (1.02)	1.53 (1.07)	1.32 (1.05)

Table 4. Average Code and Concept Scores by Task

Because performance metrics are computed for each participant by task, we also computed aggregate information seeking metrics corresponding to overall behavior levels for each participant on a given task. We computed the number of total browse sessions per task along with their corresponding number of results viewed and successful browse sessions. For search behaviors, we computed the total number of sessions per task, the number of results viewed on that task, the number of successful search sessions, and the number of sessions whose query strings were categorized as code, natural language, or hybrid. Spearman correlation coefficients for this data and the performance metrics are provided in Table 5.

	code score $(n = 76)$	**concept score** $(n = 95)$
Search Metrics		
num sessions	-0.042	0.243*
results viewed	-0.007	0.274**
num successes	0.376**	0.341**
code queries	-0.002	-0.028
nat lang queries	-0.002	0.271**
hybrid queries	-0.086	0.183
Browse Metrics		
num sessions	-0.123	0.026
results viewed	0.091	0.012
num successes	-0.095	0.010

* Correlation significant at the $\alpha = 0.05$ level.
** Correlation significant at the $\alpha = 0.01$ level.

Table 5. Correlations Between Performance and Information Seeking

Metrics for browsing activities showed no significant correlations with either performance metric. However, given that less than 4% of all browse sessions ended in an observable change to participants' code (ie., those classified as "successful"), perhaps the lack of meaningful correlations was to be expected. Further, given that 90% of browse events took place inside the object-model viewer (Adobe's API exploration tool), these results seem to suggest that participants had considerable difficulty extracting actionable information from this resource. Perhaps with additional exposure to and use of the tool, participants could improve their ability to learn from what they find. However, the apparent steepness of the initial learning curve may prove discouraging and thus deter users from referring to the API tool in the future.

Search behaviors exhibited a number of weak, but statistically significant, correlations with performance metrics. The number of instances where a search session ended in an observable alteration to the participant's source code was positively correlated with the code assessment score. While we did not analyze what was changed as a result of the search session, it stands to reason that when participants were able to identify something of relevance within the search results and attempt to incorporate it into their own project, they did better. Unfortunately as shown earlier, the overall success rate of search-based foraging sessions was less than 13%. The success rate also correlated positively to scores on the conceptual responses written in English. Additionally, positive correlations were noted for general measures of search activity level including the number of search sessions generated and number of results viewed.

Interestingly, we noted a significant positive correlation between concept score and the number of natural language queries made, but not for the number of queries containing JavaScript code. As discussed earlier, there are many difficulties when attempting to use source code as input to general purpose search engines like Google. By contrast, participants who formulated more queries using only English showed increased success rates in the foraging sessions (Spearman's $\rho(114) = 0.359, p < 0.01$). In fact, use of natural language only queries was also strongly correlated with the number of search sessions ($\rho(114) = 0.715, p < 0.01$) and the number of search results viewed ($\rho(114) = 0.691, p < 0.01$) during the various tasks.

Study Limitations

Though these results are interesting, they should be understood in the context of this study and its potential limitations. Most notably, while we endeavored to construct tasks and an environment that closely approximates the real world conditions of our end-user programmers, the artificial lab setting may have impacted participant behaviors nonetheless. For example, disallowing access to participants' friends and colleagues prohibited the potential for information gathering through their social network. Complementing our results with additional data collected in the field would help determine their generality. Further, as with most correlative work we cannot definitively determine causation from these results alone—e.g., did people perform better on the tasks because of increased success at searching, or were those with more productive search behaviors also more skilled at the tasks to begin with? Perhaps more expert participants were more aware

of when they needed to look something up and knew how to ask the question better. Follow-up studies that independently measure variables like participants' prior knowledge of programming and level of searching skill would help mitigate these confounds.

Nonetheless, these results demonstrate a relationship between use of natural language queries, overall foraging activity levels, and performance on typical tasks with which end-user programmers engage. Finding ways to encourage end-user programmers to use natural language queries within general-purpose search engines might lead to increased efficiency in locating useful resources online. We discuss this further, along with other implications of our findings, in the final section.

DISCUSSION AND IMPLICATIONS

Overall participants' information foraging behaviors are indicative of the difficulties end-user programmers face when searching for answers to code related problems. In this section we highlight some of our findings in the context of related work and discuss implications for future information resource designers.

Issues with the API Documentation

Browsing activities in both the built-in API documentation and on the Web generally took a back seat to text search activities. The majority of browse behaviors that did occur happened within the Object-Model-Viewer, Adobe's interactive API documentation tool. However, browse behaviors were not correlated meaningfully with any of the task outcome variables. Thus, even the successful participants may not have been able to effectively use the API tool as it was intended. It may also be the case that participants misunderstood the purpose of the API tool and the information it contained. We observed some participants browsing this reference for issues of programming language syntax (ie., how to write a `for` loop in JavaScript) which it did not contain. While API-style documentation listing the various packages, objects, and methods in an environment is important for professional programmers with well-defined mental models of object-oriented programming, it is less clear that end-user programmers are able to utilize such resources effectively.

Given that novices often focus on surface-level details of a task (Bransford et al., 2000), it is plausible that participants' problem solving by pattern matching approach falls short in documentation that focuses on the available abstractions in the programming environment. Prior work has shown that end-user programmers have a strong preference for example-oriented documentation (Dorn & Guzdial, 2006, 2010; Rosson et al., 2005), and we believe there is an opportunity to study new avenues to integrate mechanisms that better link and cross-index code examples (and the code therein) to the abstract principles which they embody.

The Need for Scaffolded Search Tools

Even though our participants made use of search queries for the majority of their information seeking needs, it is unclear whether they were any more successful with searching than browsing. Participants viewed fewer than one result per query event on average, they rarely refined their queries to improve the results, and only about 5% of all query events were followed by a substantive change to the participants' source code. The majority of search query strings included code (either entirely or in part) which was often copy/pasted directly from the project—another instance of participants' surface level search strategies. Unfortunately, there was no correlation between code and hybrid queries and overall success metrics. The only form of queries shown to correlate with performance metrics was the use of natural language queries.

This is not entirely surprising since participants almost exclusively used Google to perform searches, and ignored more code-oriented search tools on the Web like those within Koders, Stack Overflow, or the Adobe Exchange community. While there have been a number of recent efforts to increase the reusability of online code examples for end users (see, e.g., Gross & Kelleher, 2010b; Wightman, Ye, Brandt, & Vertegaal, 2012; Brandt, Guo, Lewenstein, Dontcheva, & Klemmer, 2009; Brandt, Dontcheva, Weskamp, & Klemmer, 2010), some of these approaches rely on contextual details automatically extracted from the programming environment. If users still rely on their everyday information foraging habits, they may be more likely to just use a general purpose search engine when given the choice, as we saw here. Unfortunately, such search engines are not designed to excel when source code makes up the query string.

One challenge that arises when code is pasted into a search box is that its language may be ambiguous. Many modern programming and scripting languages share the same basic grammar (e.g., JavaScript, C, Java, PHP), but particular small differences between them could stop a novice in his/her tracks. We analyzed query strings from Web-based searches generated in this study and found that 71.9% (140/195) of them contained an attempt to qualify the search with the name of a programming language embedded in the search text. Unfortunately, half of the participants made at least one erroneous qualification, most often using "Java" rather than "JavaScript". In these cases, we observed participants considering several irrelevant results that included code that had no chance to accomplish the task at hand. While some participants eventually corrected the query subterm, others continued to include "Java" seemingly unaware of the difference.

We propose a new class of *scaffolded search tools* that use procedural facilitation (Scardamalia, Bereiter, McLean, Swallow, & Woodruff, 1989) to help guide users through a series of query refinements. In particular, search engines could employ textual analysis techniques to identify the potential presence of code fragments within search queries (e.g., presence of camel case compound words, use of programming language keywords). If a user incorporates such terms, the search engine could present a multi-step refinement wizard that asks the user to help improve the quality of the results. Relevant questions might include: "What language are you writing code in (with an accompanying drop-down selection box)?" and "Using English, tell me a little about what you're trying to accomplish?" Such systems could also scaffold the

user in knowledge development about terminology in computer science. For example, if the search query string includes keywords like `for` or `if` it could suggest that using terms like "definite loops" or "selection" might improve the results.

While one might argue that such an approach may seem heavy-handed, users of popular search engines today are familiar with similar features. For example, spelling auto-correction in search engines and faceted refinement/navigation on many e-commerce sites. Further this approach provides a natural vehicle to limit the result search space by language and incorporate more natural language terms which could lead to more successful foraging activities (and ultimately increase efficiency and effectiveness in the end-user programming task).

Enhanced Tutorials through Case-Based Learning Aids

Lastly, a recurring issue is that novices simply lack an awareness of the scope of that which they do not yet know. In very difficult tasks that necessitate new knowledge, like the final recursion task used in this study, end-user programmers may assume they understand the scope of the problem and just provide a naïve solution with latent bugs. Providing opportunities and incentives to learn new concepts is difficult for end-user programmers, as there are competing concerns at play when they write code (Blackwell, 2002). They are not likely to enroll in formal classroom experiences, yet the knowledge gaps can hinder their ability to produce error-free solutions to their problems. A potential solution to this problem may take the form of domain-specific *case-based learning aids* (Kolodner, Dorn, Thomas, & Guzdial, 2012). Each example project (ie., case) in such a library presents a solution to a realistic programming task for the target domain. Central to each case is a narrative description of the process by which a third party solved the task—that is, it contains both the ultimate solution to the problem *and* intermediate solutions explored along the way. Such case libraries would provide collections of code examples annotated with explanatory comments that aim to build knowledge about fundamental computing concepts. An initial study of this approach with end-user programmers showed that it can indeed lead to conceptual knowledge growth while carrying out typical scripting tasks (Dorn, 2010). These learning aids would provide a powerful destination for end-users' information seeking as they provide both concrete code to be examined and borrowed and conceptual explanations that help them overcome fundamental information barriers.

ACKNOWLEDGMENTS

We thank our participants for volunteering their time to take part in this study. Data collection efforts for work were supported in part by the National Science Foundation under Grant Nos. 613738 and 0618674. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

References

Anderson, L. W., et al. (Eds.). (2001). *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives* (abridged ed.). New York, NY: Longman.

Aula, A., Khan, R., & Guan, Z. (2010). How does search behavior change as search becomes more difficult? In *Procedings of CHI'10* (pp. 35–44).

Bajracharya, S. K., & Lopes, C. V. (2012). Analyzing and mining a code search engine usage log. *Empirical Software Engineering*, *17*, 424–466.

Beringer, J. (2004). Reducing expertise tension. *Communications of the ACM*, *47*(9), 39–40.

Blackwell, A. F. (2002). First steps in programming: a rationale for attention investment models. In *Proceedings of the 2002 IEEE symposium on human centric computing languages and environments* (pp. 2–10).

Brandt, J., Dontcheva, M., Weskamp, M., & Klemmer, S. R. (2010). Example-centric programming: integrating web search into the development environment. In *CHI'10: Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 513–522).

Brandt, J., Guo, P. J., Lewenstein, J., Dontcheva, M., & Klemmer, S. R. (2009, September). Opportunistic programming: Writing code to prototype, ideate, and discover. *IEEE Software*, *26*(5), 18–24.

Brandt, J., Guo, P. J., Lewenstein, J., Dontcheva, M., & Klemmer, S. R. (2009). Two studies of opportunistic programming: interleaving web foraging, learning, and writing code. In *CHI'09: Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 1589–1598).

Bransford, J. D., Brown, A. L., & Cocking, R. R. (2000). *How people learn: Brain, mind, experience, and school* (Expanded ed.). Washington, D.C.: National Academy Press.

Burnett, M., Atwood, J., Djang, R. W., Gottfried, H., Reichwein, J., & Yang, S. (2001). Forms/3: a first-order visual language to explore the boundaries of the spreadsheet paradigm. *Journal of Functional Programming*, *11*(2), 155–206.

Burnett, M., Cook, C., Pendse, O., Rothermel, G., Summet, J., & Wallace, C. (2003). End-user software engineering with assertions in the spreadsheet paradigm. In *Proceedings of the 25th international conference on software engineering (ICSE'03)* (pp. 93–103).

Dorn, B. (2010). *A case-based approach for supporting the informal computing education of end-user programmers*. Unpublished doctoral dissertation, Georgia Institute of Technology.

Dorn, B. (2011). ScriptABLE: supporting informal learning with cases. In *ICER'11: proceedings of the seventh international workshop on computing education research* (pp. 69–76).

Dorn, B., & Guzdial, M. (2006). Graphic designers who program as informal computer science learners. In *ICER '06: Proceedings of the 2nd International Workshop on Computing Education Research* (pp. 127–134).

Dorn, B., & Guzdial, M. (2010). Learning on the job: Characterizing the programming knowledge and learning strategies of web designers. In *CHI '10: Proceedings of the 28th international conference on human factors in computing systems* (pp. 703–712).

Du Boulay, B. (1989). Some difficulties of learning to program. In E. Soloway & J. Spohrer (Eds.), *Studying the novice programmer* (pp. 283–299). Hillsdale, NJ: Lawrence Erlbaum Associates.

Duggan, G. B., & Payne, S. J. (2010). Knowledge in the head and on the web: Using topic expertise to aid search. In *Proceedings of CHI'08* (pp. 39–48).

Green, T. R., & Payne, S. J. (1984, October). Organization and learnability in computer languages. *Int. J. Man-Mach. Stud.*, *21*(1), 7–18.

Gross, P., & Kelleher, C. (2010a). Non-programmers identifying functionality in unfamiliar code: strategies and barriers. *Journal of Visual Languages & Computing*, *21*(5), 263–276.

Gross, P., & Kelleher, C. (2010b). Toward transforming freely available source code into usable learning materials for end-users. In *Evaluation and usability of programming languages and tools* (pp. 6:1–6:6). New York, NY, USA: ACM.

Gwizdka, J., & Spence, I. (2006). What can searching behavior tell us about the difficulty of information tasks? a study of web navigation. In *Proceedings of ASIS&T'06* (pp. 1–22).

Ko, A. J., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M., et al. (2011). The state of the art in end-user software engineering. *ACM Computing Surveys (CSUR)*, *43*(3), 21.

Ko, A. J., Myers, B. A., & Aung, H. H. (2004). Six learning barriers in end-user programming systems. In *VL/HCC '04: Proceedings of the 2004 IEEE symposium on visual languages and human-centric computing* (pp. 199–206).

Ko, A. J., Myers, B. A., Coblenz, M. J., & Aung, H. H. (2006). An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *Software Engineering, IEEE Transactions on*, *32*(12), 971–987.

Kolodner, J. L., Dorn, B., Thomas, J. O., & Guzdial, M. (2012). Theoretical foundations of learning environments. In (2nd ed., pp. 142–170). New York: Routledge.

LaToza, T. D., Garlan, D., Herbsleb, J. D., & Myers, B. A. (2007). Program comprehension as fact finding. In *Proceedings of ESEC–FSE'07* (pp. 361–370).

Leshed, G., Haber, E. M., Matthews, T., & Lau, T. (2008). Coscripter: automating & sharing how-to knowledge in the enterprise. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on human factors in computing systems* (pp. 1719–1728).

Liu, C., Liu, J., Cole, M., Belkin, N. J., & Zhang, X. (2012). Task difficulty and domain knowledge effects on information search behaviors. In *ASIS&T'12: proceedings of the 75th annual meeting of the association for information science and technology* (p. n.p.).

Liu, J., Gwizdka, J., Liu, C., & Belkin, N. J. (2010). Predicting task difficulty for different task types. In *Proceedings of ASIS&T'10* (p. n.p.).

Nardi, B. A. (1993). *A small matter of programming: Perspectives on end user computing*. Cambridge, MA: MIT Press.

Pane, J. F., Myers, B. A., & Miller, L. B. (2002). Using HCI techniques to design a more usable programming system. In *Proceedings of the 2002 IEEE symposia on human centric computing languages and environments* (pp. 198–206).

Rosson, M. B., Ballin, J., & Rode, J. (2005). Who, what, and how: A survey of informal and professional web developers. In *VL/HCC '05: Proceedings of the 2005 IEEE symposium on visual languages and human-centric computing* (pp. 199–206).

Scaffidi, C., Shaw, M., & Myers, B. (2005). An approach for categorizing end user programmers to guide software engineering research. In *Weuse i: Proceedings of the first workshop on end-user software engineering* (pp. 1–5). New York, NY, USA: ACM Press.

Scardamalia, M., Bereiter, C., McLean, R. S., Swallow, J., & Woodruff, E. (1989). Computer-supported intentional learning environments. *Journal of educational computing research*, *5*(1), 51–68.

Segal, J. (2007). Some problems of professional end user developers. In *VL/HCC'07: Proceedings of the 2007 IEEE symposium on visual languages and human-centric computing* (pp. 111–118).

Spohrer, J., & Soloway, E. (1985, November). Putting it all together is hard for novice programmers. In *Proceedings of the IEEE international conference on systems, man, and cybernetics.*

Teevan, J. (2008, October). How people recall, recognize, and reuse search results. *ACM Trans. Inf. Syst.*, *26*(4), 19:1–19:27.

White, R. W., Dumais, S. T., & Teevan, J. (2009). Characterizing the influence of domain expertise on web search behavior. In *Proccedings of WSDM'09* (pp. 132–141).

Wiedenbeck, S. (1988). Learning recursion as a concept and as a programming technique. In *SIGCSE '88: Proceedings of the nineteenth SIGCSE technical symposium on computer science education* (pp. 275–278).

Wightman, D., Ye, Z., Brandt, J., & Vertegaal, R. (2012). Snipmatch: using source code context to enhance snippet retrieval and parameterization. In *Proceedings of the 25th annual acm symposium on user interface software and technology* (pp. 219–228).

Wildemuth, B. M. (2004). The effects of domain knowledge on search tactic formulation. *JASIST*, *55*(3), 246–258.

Wilson, A., Burnett, M., Beckwith, L., Granatir, O., Casburn, L., Cook, C., et al. (2003). Harnessing curiosity to increase correctness in end-user programming. In *CHI '03: Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 305–312).