Foraging among an Overabundance of Similar Variants

Sruti Srinivasa Ragavan[1], Sandeep Kaur Kuttal[1,2], Charles Hill[1], Anita Sarma[1], David Piorkowski[1], Margaret Burnett[1]
[1]Oregon State University and [2]University of Tulsa
[1]Corvallis, OR, and [2]Tulsa, OK, USA
{srinivas, hillc, anita.sarma, piorkoda, burnett}@eecs.oregonstate.edu, sandeep-kuttal@utulsa.edu

ABSTRACT
Foraging among too many variants of the same artifact can be problematic when many of these variants are similar. This situation, which is largely overlooked in the literature, is commonplace in several types of creative tasks, one of which is exploratory programming. In this paper, we investigate how novice programmers forage through similar variants. Based on our results, we propose a refinement to Information Foraging Theory (IFT) to include constructs about variation foraging behavior, and propose refinements to computational models of IFT to better account for foraging among variants.

Author Keywords
Reuse, variants, Information Foraging theory.

ACM Classification Keywords
H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous

INTRODUCTION
Computer-supported creative tasks—such as writing, graphic design, creating presentations, and some forms of programming—are often exploratory in nature. People often work on such tasks in an opportunistic manner: integrating bits and pieces from various sources, evaluating various alternatives, etc. During this process, they also save their intermediate steps, thereby creating several variants of the same artifact [2, 12, 43].

When performing creative tasks, in addition to reusing bits and pieces from a variety of sources, individuals may also revisit earlier variants of the same source. This especially occurs when things go wrong, they reach a dead-end, or when they want to use features from other (earlier) variants [2, 12, 17].

One example of such creative tasks is exploratory programming, in which programmers experimentally blend creating new code with reusing bits and pieces of code from various sources, some of which may be earlier variants. Both novice and expert programmers engage in such exploratory programming [17].

In this paper, we focus on how programmers reuse variants in exploratory programming [1, 27]. Reuse in programming is generally difficult [14], and becomes even more difficult in the context of reusing earlier variants. This is because temporally close variants of the same code can be very similar, requiring much cognitive effort to choose which variant to reuse.

This cognitive effort is especially high in programmers with little experience in systematic software reuse [29]—such as novice programmers. Novices do tend to engage in exploratory programming with numerous variants in the form of alternatives they're trying. These factors can inhibit novices' explorations during programming [45]. Given these challenges, in this paper, we focus on novice programmers.

We believe that theories on information seeking behavior can help understand how novice programmers search for code among variants. Information Foraging Theory (IFT) is one such theory that has explained information-seeking behavior of people in general, and of programmers in particular [39]. IFT proposes that a person seeking information follows the "information scent" from sources (patches) that exist in an environment, similar to the way predatory animals in the wild follow the scent to their prey [38]. Models based on the theory have accurately predicted the links that people follow as they navigate through web sites and software artifacts [24, 32, 38].

However, we posit that IFT is underexplored in the space of exploratory programming. Specifically, IFT has not yet dealt with foraging in the presence of variants over time—where multiple artifacts have close similarities, but also some differences. We propose a refinement of IFT as a theory of variation foraging, modified to account for how people "forage" through variants. Abstracting how people search through variants via theory can provide a foundation that can help the design of environments for creative tasks, such as programming.

In this paper, we take the first step towards a theory of variation foraging through a qualitative, empirical study investigating how novice programmers find and evaluate variants. We structured our study around the following research questions:

RQ1: What are the types of information that help novice

CHI'16, May 07-12, 2016, San Jose, CA, USA
 ISBN 978-1-4503-3362-7/16/05···$15.00
DOI: http://dx.doi.org/10.1145/2858036.2858469

programmers identify variants that can be reused?

RQ1a (*Between-variant foraging*): How do they forage between variants of an artifact to find and evaluate a potential variant?

RQ1b (*Within-variant foraging*): How do they forage within a specific variant to find and evaluate a potential patch?

RQ2: How do novice programmers integrate parts of the program across variants?

BACKGROUND AND RELATED WORK

Variations

Background

To understand how novice programmers forage through variants, we first define program variants. From a given starting point, a program (which may be empty) can be modified into other programs through the addition, removal, or other modification of code. Informally, we use the term *variation* with respect to software when multiple related implementations exist, either serially or in parallel. We use the word *variant* to more specifically refer to a syntactically valid program that occurs together with similar, related programs in a group. For example, if a user edits an application to use a menu instead of buttons, this results in a new variant of the application.

Related Work

Research has explored providing variation support in non-programming domains, such as graphic design, documents and personal information management. Several tools support variations in graphics and interface design by allowing users to create multiple options, and providing ways to manipulate and compare them [12, 13, 43, 44]. For example, Kumar et al. [18] present Bricolage, which helps web interface designers transfer the style and layout of one web page to another. Mechanisms to automatically support variations have also been proposed for personal information management. Karlson et al. [16] introduce the concept of copy aware computing ecosystems that track user edits in a computing environment. They show that such personal information management systems can help users keep track of changes and semantics behind their copy operations.

In professional software engineering, product lines [6] refer to a family of customizable software products created through a rigorous configuration process. Research on version control systems, in both academia and industry, has focused on supporting and managing variations to programs. There also exists research on providing variation support to end-user programmers [19, 21].

Our work differs from the above work by studying how novice programmers navigate, find, understand, and reuse code between different variations of the same set of programs, when performing exploratory programming.

Information Foraging Theory

Background

We use information foraging theory to inform our understanding of *how* and *where* novice programmers forage when reusing program variants. Information foraging theory (IFT), developed by Pirolli and Card [39] has been used to understand how humans search for information, and is based on optimal foraging theory.

Optimal foraging theory, rooted in the biological sciences, is a theory of how animals hunt for food. Pirolli and Card found similarities between users' information search patterns and animals' food foraging strategies, and used these similarities to develop IFT. There are several constructs in IFT that relate to optimal foraging theory: humans are *predators* who search for their *prey,* which is information. To find their prey, the predators look through various information sources called *patches*, such as a file explorer window, source code file, or an output window – see Figure 1 parts: A, B, C, respectively. Patches contain *information features*. Information features are elements of the patch, such as folder names or variable names (Figure 1 parts: D, E), which a predator can process to gain knowledge.

In each patch, predators make one of three choices: (1) to forage for information within the patch, (2) to forage to a new patch or (3) to engage in enrichment by modifying their environment. These choices are informed by the predator's *information scent*, gathered from *cues* (signposts) in the environment, such as labels on links. Thus, the scent of a cue is the predator's assessment of the value and cost of information obtained by following a link associated with that cue. The network of patches connected by links is called the *topology*.

Related Work

Information foraging theory was introduced by Pirolli and colleagues to explain people's foraging behavior in large document collections and web pages [3, 4, 11, 34, 35, 36, 37]. To assess the theory empirically, Pirolli and his colleagues built several computational models. For example,

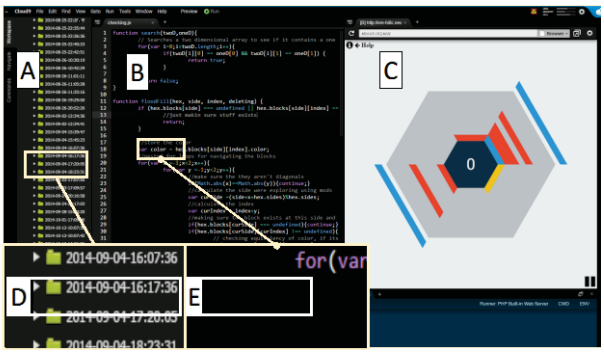

Figure 1: The cloud9 Interactive Development Environment. A, B, C are Patches; D, E are information features (see text).

Fu and Pirolli built a cognitive model called SNIF-ACT [35] that modeled the links on a page and computed scent to predict users' foraging and backtracking. Chi et al. built another model called WUFIS [5] that built a graph of all pages and links on a website to predict foraging activity and identify the least accessible pages.

IFT has also been applied to software engineering, in areas like program maintenance, debugging, and requirements [10, 22, 23, 25, 26, 28]. For example, Lawrance et al. developed a model called PFIS [24] to predict programmer navigation during maintenance tasks, and then a reactive version, PFIS2, to account for evolving foraging goals [22, 23, 26]. Piorkowski et al. used IFT as a lens to understand programmers' foraging goals and strategies during debugging [30] and introduced PFIS3 [32] and a PFIS-based tool that recommends the next navigation step a programmer should take [31].

Kuttal et al. applied IFT to understand end-user foraging behavior when debugging mashups [20], thereby refining understanding of end users' debugging behaviors. They also categorized different types of cues and strategies that users could use when debugging mashups.

All of these IFT works have focused on foraging in only *one* variant of an artifact (often the most recent one [26], even in reuse situations [10]). In contrast, this paper considers foraging in multiple variants of the same artifact that are *all* available at once. One important foraging impact of multiple variants is that these variants may be very similar.

METHODOLOGY

We conducted a think-aloud study to investigate how our target population would forage in an information space containing a large number of program variants.

We used Hextris [8], a web-based puzzle game inspired by Tetris, as our test environment. We obtained the game's source from a public GitHub repository [9]. Hextris fulfilled our criterion of containing a large number of variants (over 700 commits) that the participants could potentially use in order to complete their tasks.

We observed participants while they performed tasks. We used screen-capture software to capture and record their actions on screen, and a webcam to capture video and audio of the participants directly.

Task Context

Participants were presented with a scenario in which a small non-profit company hosted the Hextris game on their website. In this scenario, volunteer programmers helped the company improve Hextris over its lifecycle and visitors to the site suggested some changes to the game. We asked participants to implement these changes.

Tasks

The first task was to move the game's score indicator from the center of the hexagon (see Figure 2) to a location above the hexagon like it had been in earlier variants of the game (see Figure 3). We used the phrase "like it was before" when communicating this to the participants, both to phrase the statement as a comment from a site visitor and to avoid explicitly mentioning that a solution existed in an earlier variant. The second task was to return the game's bonus score multiplier indicator (see Figure 2) to the location above the hexagon and put it in parentheses, "like it was before" (see Figure 3). The third task was to change the text color of the score and multiplier to black so it could be seen when placed above the hexagon. Full task descriptions are available on our supplementary website [42].

Several earlier variants of the game had the score and the multiplier above the hexagon; therefore, multiple solutions to the tasks were possible. Some of these earlier variants had score calculation logic similar to the current one while others had an entirely different logic. We asked participants to preserve the current score calculation logic.

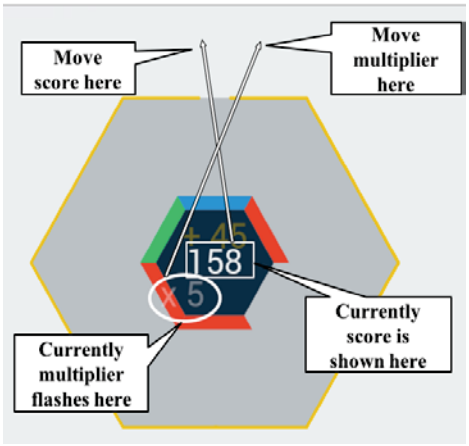

Figure 2: Participants were asked to make the following changes to the latest variant of the game: i) move the score (boxed) above the hexagon, ii) move the bonus multiplier (circled) above the hexagon and iii) change the color of the score and multiplier to black so it could be seen when placed above the hexagon.

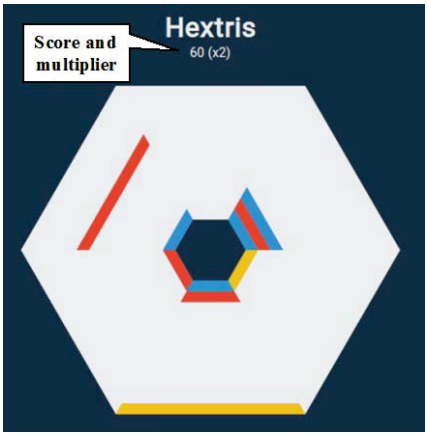

Figure 3: In some earlier variants of the game, the score indicator and bonus score multiplier appeared above the hexagon. This image was not provided to participants.

Participants

We were interested in how programmers navigate through variants during exploratory programming. Since novice programmers have been known to engage in exploratory programming [1], we recruited Computer Science students from our university. All participants had some experience with programming, but were relatively new to JavaScript programming (less than two years). There was one outlier who indicated that he had 6 years of JavaScript experience, but he mentioned that he'd only occasionally programmed with JavaScript. Participant ages ranged between 18 and 29. Table 1 shows the general demographics of the participants.

Study Design

The participants first filled out a background questionnaire, and then a researcher instructed them in the think-aloud method. Participants were then given a short tutorial on how to use Cloud9 [7], a web-based JavaScript IDE. The participants who lacked JavaScript or web development experience were also given a 15-minute tutorial on the basics of HTML, CSS, and JavaScript.

After the initial instructions, participants spent 50 minutes working on the three programming tasks, talking aloud as they worked. Throughout the session, we collected audio of what the participant said, video of the participant while talking aloud, and screen-capture video. Following a short break, we conducted a semi-structured retrospective interview with each participant by playing back all of the screen-capture video. After each foraging decision we observed, we stopped the video and asked the participant questions about their foraging decision. Interview questions were inspired by work by Piorkowski et al. regarding IFT in debugging [33], and are detailed in Figure 4.

Presentation of Variations

Over 700 program variants were available to the participant. We labeled each variant by the timestamp of the commit. We also included a file called changelog.txt within each variant folder that contained the GitHub commit message and other information (e.g., commit ID, author name). Our participants had different levels of experience with GitHub. To control for this experience level, we presented variants in the Cloud9 environment, which none of our participants had used. Figure 5 shows how variants were displayed in the explorer view in Cloud9.

Limitations

Every study has limitations. In our study, the experimental setup and the presentation of variations as separate folders were different from full-fledged version control systems that professional programmers use. Similarly, the tasks and the Github repository used may not be representative. Programmers may also have different motivations for foraging than those used in our study. Limitations like these can only be addressed through further empirical studies.

Qualitative Analysis

We qualitatively coded the data as follows: We used the baseline code set inspired by previous research [21, 33, 41]. We also added new codes (shaded items in Table 2) to record phenomena that we observed specific to our variation-foraging questions.

We segmented the transcripts of participants' think-aloud videos into 30-second segments and coded them, allowing multiple codes per segment. Two researchers independently coded 20% of the transcripts. We then calculated inter-rater reliability using the Jaccard measure [15], resulting in a rate of agreement of 85% on 20% of the data. Given this high rate of inter-rater reliability, the two researchers split up the coding of the remaining code set.

Participant label	Gender	Level	Age	Experience (years)		
				Java-script	Programming	Web development
P01	Male	Sophomore	20s	1	1.5	Yes
P02	Male	Freshman	Teens	6*	6	Yes
P03	Female	Junior	20s	0	9	No
P04	Male	Sophomore	20s	2	1	Yes
P05	Male	Freshman	Teens	0	3	No
P06	Male	Sophomore	Teens	0	5	Yes
P07	Male	Junior	20s	2	2	Yes
P08	Male	Junior	20s	1	5	Yes

Table 1. General demographics.
(*Participant P02 occasionally programmed in JavaScript)

Each time a participant entered a new variant:
Explain:
You chose to [do/go to] (Variant name)
Ask:
What did you expect to (see/find) when you went to ____?
What did you see as your other possible choices?
Why did you choose to navigate to ____ as opposed to (other choices)?

Figure 4: During the retrospective interview, we played back a video of the participant's programming task and asked these questions after every foraging decision.

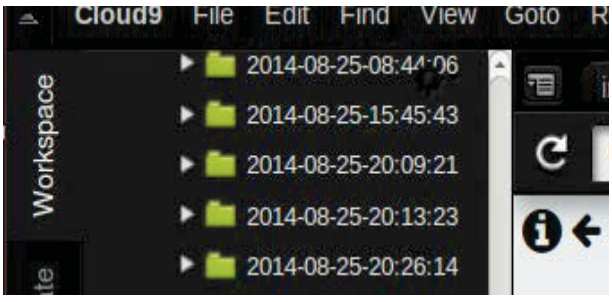

Figure 5: Variants were presented in chronological order in the IDE.

Code	Description
Cue-Types	
Create Time, Update Time	Timestamp cues marking latest, first or intermediate variants, and navigation to corresponding variants.
Previous File, Previous Method	Reuse of information features (file and method names) from one variant as cues in other variant.
Output	Cues based on how output looks or running a preview
Domain	Game-related words, e.g., score, block, etc
Source	Source Code-Inspired cues e.g., function name, variable name, etc.
Error, Correct	Cues based on error/correctness of patch/prey
File Name, File Type	FileName-Inspired and file type cues
Document	Documentation cues: change logs, readme files, tooltips, etc.
Comment	Source code comments
Search	Search inside IDE or the internet
Debug, Inspect	Debugger or "element inspect" feature in browser
Operations	
Edit	Edits made to source code, to verify the prey using Output-Inspired cues, or to implement the task.
Reuse	Explicit reuse of source code , i.e., copy and paste
Compare	Compare two variants
Navigations	
Between Variant Navigation	Between-variant navigation was coded along with the cues that guided these navigations.

Table 2. We coded participants' operations, navigations and the cue-types they attended to. The highlighted items are the new codes we added pertinent to variations foraging that have not been reported in the IFT literature.

We coded the data according to the cue types participants used, the type of operations they performed, and their navigation behavior (see Table 2).

RESULTS

Rosson & Carroll [40] studied how Smalltalk programmers find and apply code from one "usage context" (containing reusable code) to accomplish a task in their "current context" (containing the location to integrate the reusable code). They define three stages for reuse tasks:

1) Finding a usage context,
2) Evaluating a usage context,
3) Debugging a usage context

This model was for reuse for a single variant of source code, so we extended the model in two ways to accommodate multiple variants. First, we introduce a "Finding and evaluating a current context" stage, because in our case this stage influenced how participants found and evaluated the usage context. Second, participants integrated changes across variants to finish the task. Thus, we replace the debugging stage with an "Integrating the variants" stage. The modified reuse model consists of the following stages:

1) Finding and evaluating a current context,
2) Finding and evaluating a usage context,
3) Integrating the variants.

In IFT terminology, a current or usage context maps to the patches within a variant that are relevant to the user's ongoing task (see Figure 6). Here, we term the current context as destination variants and patches, and usage contexts as source variants and patches. Therefore, in order to find a context, one has to forage (find and evaluate) the right vari-

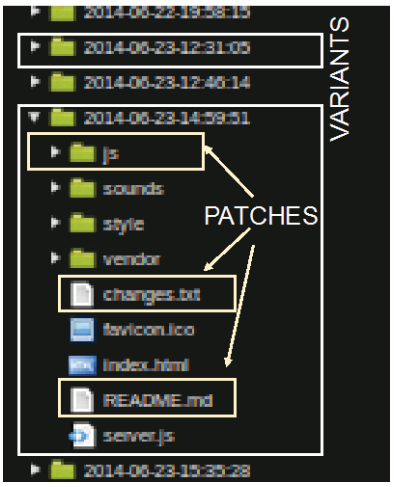

Figure 6: Finding and evaluating a context involves finding the variant (between-variant foraging) and then the patches within the variant (within-variant foraging).

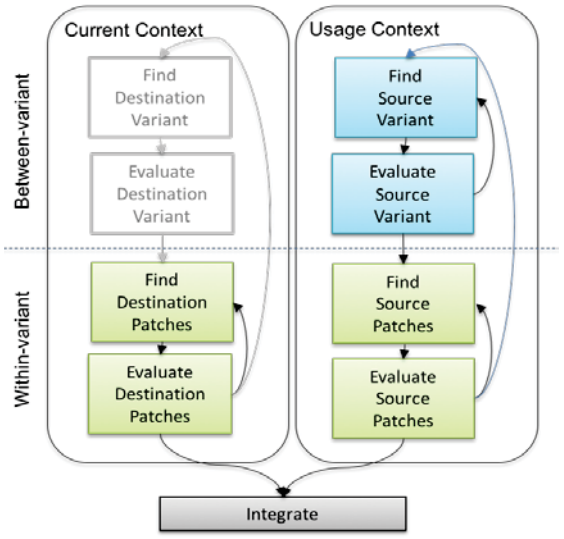

Figure 7: Modified reuse model: Participants were provided with the destination variant (greyed out). They interleaved finding and evaluating the prey in both between-variant (blue) and within-variant (green) foraging.

ant, and then forage (find and evaluate) the relevant patches inside that variant (see Figure 7).

Note that we do not intend our modified reuse model to suggest that programmers followed any particular order. In fact, Figure 8 shows that Participant P06 foraged for the source variant and the source patches (usage context) before he foraged for the destination patches (current context). In summary, our results describe *sets*—not sequences—of behaviors.

Stage 1: Finding and evaluating a current context

Foraging for the current context, or where the fix needed to be made, involved finding the right destination variant, and then finding the relevant destination patches in that variant.

Destination Variant: Find and Evaluate

Participants didn't need to forage for the destination variant: it was provided for them and labeled "Current".

Destination patch: Find and Evaluate

Finding and evaluating patches were interleaved when participants foraged for the destination patches. In conformance with prior research on IFT for debugging, participants attended to various cues within the destination variant (e.g., file names, words in source code, source code comments, Output-Inspired, etc.) [33].

When participants found a patch that might contain the prey, they edited the code and used the resulting output to evaluate it. For example, all participants altered the `x` and `y` parameters in the `renderText()` method in order to evaluate whether it really altered the score position. The dots inside the destination patch foraging segments show how participants edited the destination patches to evaluate them.

Stage 2: Finding and evaluating a usage context

For Stage 2, we investigated the types of information that participants used to identify reusable variants (RQ1), their between-variant foraging behavior (RQ1a), and their within-variant foraging behavior (RQ1b).

Source variant: Find

Participants first tried to find a variant of the game that had the score and multiplier above the hexagon. To do so, they foraged among several variants. We call this *between-variant foraging*, similar to between-patch foraging [39].

Indeed, when foraging within any one variant (within-variant foraging), participants performed both within-patch and between-patch foraging. However, their between-variant foraging was not similar to between-patch foraging; the latter refers to navigating between patches that have *different* information, e.g., between two different methods, but the variants were more *similar* than different.

The only cue type available to participants while foraging between variants was the update timestamp of the variant (specified as the folder label). Five out of eight participants attended to the timestamp cues by navigating to the oldest variant of the game. Participant P04 said: "*I wanted to see what they were doing at the beginning, [to] see what they had implemented to start off with*". Participants then navigated to a variant by either directly selecting a particular timestamp, or by using the chronological ordering to guess how far to scroll to a potentially more valuable variant.

Participants' timestamp-based navigation mostly followed one or more of the following three patterns, shown in Figure 9. Note that regardless of the foraging pattern they followed, all participants skipped over several variants since consecutive variants were too similar.

1) Unidirectional: Four out of eight participants (P01, P02, P03, P04) foraged in a single direction. They either foraged from the oldest to the most recent variant or vice-versa (see Figure 9 (a)). Participant P03 explained: "*... jumping down more ... like a sorting algorithm, checking further and further until there was a change.*"

2) Bidirectional: Four out of eight participants (P03, P06, P07, P08) changed directions while foraging between can-

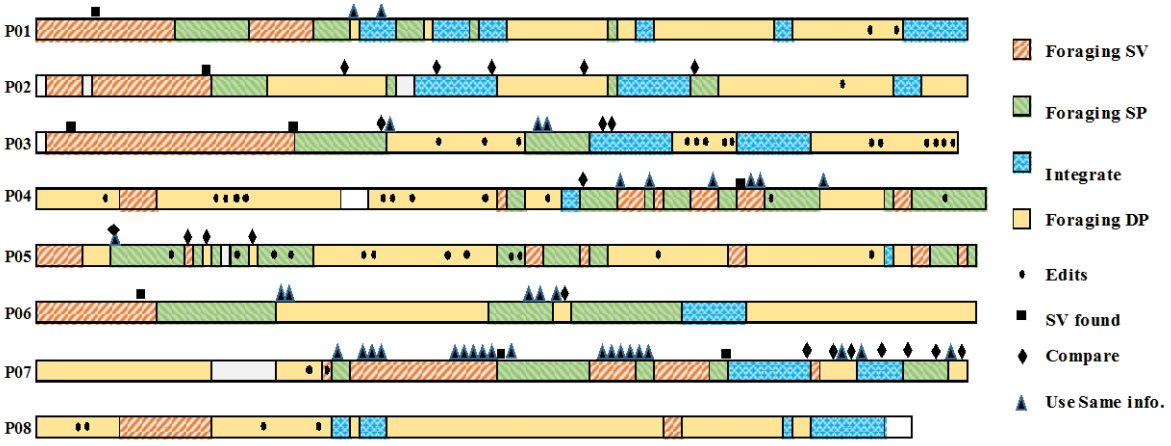

Figure 8: Participants foraged the Source Variant (SV), Source Patches (SP) and Destination Patches (DP) in different orders. Note that they used information features (shown as triangles) from one variant to forage in other variants and performed comparison during both source-patch foraging and destination-patch foraging.

didate source variants. Initially, they started from either the oldest or the most recent variant, and foraged along one direction. However, when they found that they had gone too far in one direction, they reversed course and continued in the other direction, as shown in Figure 9 (b).

3) Systematic narrowing: Two of the eight participants (P02, P05) started from a variant in the middle and systematically narrowed down the search space using an approach similar to a binary search. Participant P02 said: "*... just split the list in half and then ... do a binary search on it.*"

Besides attending to the timestamp cues, all participants enriched the environment. Some participants kept variant folders expanded so they could quickly identify previously-visited locations, such as Participant P03: "*...having seen an open folder ... I figured that is the one that I saw earlier."* Further, half of the participants collapsed variant folders that did not contain relevant prey before moving on to the next variant.

Participants also kept track of variants by remembering their timestamps. For example, P01 said, *"So I'll just remember that on 2014-05-20, [it] had the right interface."*

Source variant: Evaluate

Participants went through several *find-evaluate* cycles while foraging for a source variant, to evaluate whether the variants contained relevant information features. If a participant found a variant to be relevant, they then foraged more within the same variant; we call this type of foraging *within-variant foraging*. Otherwise, they looked for another source variant; we call this *between-variant foraging*.

Figure 7 shows how the evaluation of a variant either led to foraging for the patches within the variant (if it was the right variant), or finding another variant (if the variant was not suitable). Most participants also continued to forage for

Figure 9: (a) Participant P01 was a unidirectional forager. (b) Participant P06 was a bidirectional forager. (c) Participant P05 was a systematic narrowing forager. Note that they all skipped a few variants at a time

alternate source variants even after they found a suitable variant (with score and multiplier above the hexagon). For example, the first square above row P07 in Figure 8 shows the first source variant he found. After briefly foraging within that variant and finding it unsuitable, he foraged for another variant (the second square above row P07). One goal of participants during such foraging was to find a closer match to the destination variant.

In order to evaluate whether a variant might contain the prey that the participant was looking for, they attended to several cue types within the variant. Table 3 shows the cue types that participants used to evaluate variants, along with the frequencies of their usage. When evaluating, participants often used these cues as signposts *away* from the prey, inferring a negative scent from these cues and then navigating directly away from the variant.

1) Output-Inspired: Output-Inspired cues were the most frequently used cue types, accounting for 68.3% of all evaluation cues. Participants' attention to this cue type is unsurprising because the task required the game to be "like it was before", and the easiest way to determine the appearance of a game was by previewing it. Participants focused on the game's features and easily rejected variants of the game where the *score* and *multiplier* features did not appear above the hexagon. They also looked at the game's correctness. Most participants rejected variants of the game that had errors or did not work. The only exception to this was Participant P07 who continued foraging inside a "broken" variant because he found the source code to be still useful: *"This drawing code, that's not actually running, still has basically the same drawScoreboard*".

2) Change Log-Inspired: Each variant folder included a change log in a file named changes.txt, as described in the methodology. Participants frequently used these cues to understand what changes were made in the variant. Participant P01 said: "*I expected to see something along the lines of 'changing the position of score'".* Change Log-Inspired cues were the second most popular cue type, accounting for 15.8% of all the cues that participants used to evaluate vari-

	Cue Types			
	Output-Inspired	Change Log-Inspired	Source Code-Inspired	FileName-Inspired
P01	29	12		
P02	15	8		
P03	44		3	
P04	6	15	2	
P05	9		2	
P06	22		1	
P07	12		22	3
P08	14		2	
Total Occurrences	151 (68.3%)	35 (15.8%)	32 (14.5%)	3 (1.4%)

Table 3. Participants used some cue types more frequently than others to evaluate variants.

ants. However, change logs were sometimes unhelpful because they were non-descriptive (e.g. one log contained only the text "asdf"). When one participant (P03) found the change logs non-descriptive, she abandoned this cue type for the rest of the session: "*Their document isn't that good for people changing things."*

3) Source Code-Inspired: The third most frequently used cues were Source Code-Inspired (14.5%). Participants like P07 used similarities and differences in source code as cues to evaluate a variant: *"[source code] this still looks like the center to me*". When he saw that the source code between two variants were similar, he perceived a negative scent, and navigated to a different variant. Some participants read the source-code to determine the game features in that variant, such as P03: *"... looking at the lines of code and determining if it had the things I needed or not."* Some participants also perceived errors in the source code (e.g., compilation errors) as a negative scent, and did not forage further in those variants.

4) FileName-Inspired: One participant (P07) used File-Name-Inspired cues for evaluating variants and rejected variants where certain files were absent. He remarked about one variant: "*... at some point, [filename] did not even exist*" and did not forage further within that variant.

Source patch: Find

Once participants found a source variant, they proceeded to forage for the source patches within that variant. They looked for the exact patches relevant to their tasks, similar to how they foraged earlier in the destination variant. Although the participants' foraging goals in the two cases were similar (e.g., find code that rendered the score), their foraging behaviors were different.

Earlier, when participants foraged within the destination variant, they did not know where relevant patches might be. They relied on cues in the environment to forage the destination patches. However, when foraging in the source variant (which was similar to destination variant), participants had formed expectations about where the source patches might be located. These expectations provided a starting point for participants' foraging within the source variant.

More specifically, participants used information features from the destination variant to directly navigate to appropriate patches in the source variant. For example, P03 found calls to the method renderText in a file named view.js in the destination variant. When she later foraged in the source variant, she said, "*renderText is still something I can look for*" and searched for "render" in the view.js file. Six out of seven participants who foraged for patches in both source and destination variants used such similarities to guide their foraging (P1, P3-P7: the blue triangles in Figure 8).

Looking for identical patches or information features across similar variants was an easy way for participants to navigate to the patches that they thought might contain relevant information. However, there were cases where this strategy failed. Participants sometimes did not find the same patches or information features that they expected to find because those patches had changed over time. In such cases, participants proceeded in one of the following two ways:

1) Participants continued to rely on the similarities between the variants, hoping that other similar information features might lead them to the right patches. For example, when Participant P03 did not find the renderText() method in the source variant, she said: *"what are some other key phrases I can look for... I guess go back and check for score."* In Figure 8, row P03, the two consecutive triangles denote how she looked for renderText and then immediately looked for score.

2) Participants changed their strategy without further reliance on the similarities between the variants. They looked for other cues within the source variant based on cues that they had found in their earlier foraging within the destination variant. For example, Participant P03 searched for renderText and score in the source variant, but when these searches did not lead her to the right patches, she started reading the source code within that variant, as shown by the absence of triangles following the two consecutive triangles in row P03 of Figure 8.

Source patch: Evaluate

Participants interleaved finding the source patches with evaluating them, mainly performing two kinds of evaluation on the source patches.

First, some participants wanted to ensure that the behavior of the source-code was what they expected. They made edits to the code and checked the effect on the output, just like they evaluated the destination patches. Two out of eight participants performed this kind of evaluation (see the edits during source-patch foraging in Figure 8). If participants found that the patch they were on was not the right patch (e.g. their edits didn't change the location of the score), participants undid their changes and continued foraging for patches within the same variant.

The other kind of source-patch evaluation was geared towards participants' eventual goal of reuse: they evaluated whether a source patch was suitable for integration with the destination patch. In order to evaluate whether a certain source patch was suitable for reuse, participants compared the information features between the source and destination patches. While similar patches led the participants to the integration phase, differences between the source and destination patches often acted as a negative scent, causing the participant to start over by looking for another source variant. This is because participants expected that the cost of integrating similar patches would be lower than integrating dissimilar patches. Participant P01 commented: *"I am going to try and find any styles that apply to score ... and copy that and put it down into here"*. All seven participants who foraged for source patches within a source variant exhibited this behavior.

While participants mostly used a drill-down approach of foraging for the variant, and then the patches within the variant, one participant (P07) evaluated source variants by evaluating the patches. In Figure 8, row P07's sequence of triangles shows how he used destination patches to evaluate source variants. This also explains his heavy usage of Source Code-Inspired and FileName-Inspired cues to evaluate variants (see Table 3).

Evolution Stories: Guiding Variation Foraging

As participants foraged between and within variants, some built a story of how the game had evolved. When participants could not build a complete story, they created a partial outline. Participants then used the evolution story to guide their foraging.

These stories were largely based on the information features that participants collected in the various patches that they had already foraged. Some used Output- and Change Log-Inspired cues to understand how the game had evolved. For example, Participant P08, who followed Output-Inspired cues, commented: "*seems that the game had the zero in the middle and then before that never worked. That could've gotten broken at some point though*". Other participants used Source Code-Inspired cues to understand how the code had evolved over time, such as when Participant P07 examined a method across variants and said: *"At some point, this [method] was [re]factored into its own function, then it was [re]factored back out of its own function"*. We think that this behavior is unique to foraging among variants.

More specifically, in the retrospective interview, P07 explained how he built, refined and used a story to guide his foraging. He first used information features in the output to build a story: *"...early in development there's no score label. At some point the original score label is introduced. And then, after that, the 2nd score label's introduced"*.

As he processed more information features from the output of more variants, he refined his story: *"... after dealing with this for a while, there might have been like no score label, then the original score label, then no score label, and then the second score label for a while"*.

He then used his story to guide his foraging: "*... that's basically why, when I hit this version that had no score label, I just decided to start searching in a more recent direction"*.

Stage 3: Integrating the variants

Once participants found and evaluated the current and usage context, they proceeded to the third stage of reuse, *integrating the variants,* to complete their task. Participants used the following strategies in integrating the changes:

1) Copy and paste: When two variants were similar, participants attempted to copy and paste the code from the source variant into the destination variant, and made minimal modifications to match the task requirement. Only one participant (P06) was able to find such a similar patch for reuse, and only for one of the tasks. In other cases, when the two variants were dissimilar, participants copied and pasted code from the source patches into destination patches, and then fixed all the dependencies and errors. Three out of eight participants (P01, P07, P06) followed this strategy.

2) Re-implement: Two participants (P07, P03) implemented the task from scratch (without reuse) when they found source and destination variants to be dissimilar. Further, one participant (P08) could not locate the right source variant; therefore, he directly implemented the fix only based on the (textual) task descriptions.

These two strategies have been reported in reuse literature as "cut-and-stanch-the-bleeding" and "analyze-then-act" strategies, respectively [14]. How participants integrated the code depended on the variant and the patches that they had foraged, and the perceived cost of integrating the changes.

DISCUSSION

Implications for Theory and its Models

Traditionally, IFT models work with collections of *dissimilar* but *connected* patches. For example, IFT models have predicted foraging behavior in web sites (groups of dissimilar but linked pages) [39] and in software source code (groups of dissimilar files that refer to each other) [24, 28].

In contrast, modeling foraging through variants must predict how people will forage through collections of *similar*, but *disconnected* patches. Our results indicated that users navigated across these disconnected collections using two main strategies. First, they focused on the differences between patches across similar variants (6 out of 8 participants). Second, they foraged across variants by following their evolution history (5 out of 8 participants). In particular, these participants generated temporal stories about how the program (and its variants) evolved over time. These stories guided their navigations and were fundamental to some participants' foraging.

These two foraging behaviors are interesting, because they may be uniquely important to variation foraging; IFT models in more traditional settings do not normally come across so much similarity. Therefore, IFT computational models in variation settings may need to be extended to accurately describe such behaviors.

Four implications for IFT go beyond computational models, and directly impact the theory itself. First, there is no construct in IFT that could be instantiated as a story: stories are not cues, not patches, not prey, and so on. Thus, an open research question is whether new IFT construct(s) are needed to capture this phenomenon.

Recent IFT research [33] has begun to explore the concept of cue types based on their provenance—a perspective on cues beyond their content. Our results suggest our second implication for theory: another new perspective, that of cues that signal differences versus those that signal similarities. In our results, 7 out of 8 participant verbalizations referred to looking for differences between variants. On the

other hand, cues that highlighted similarities between variants were also important: to ease integration of code between variants, participants compared the code structure to try to find the source variant that was the most similar to the destination variant.

Recognizing the differences and similarities between variants inherently implies that the forager is doing some kind of comparison, leading to our third implication for theory. Current IFT models treat foraging-within, foraging-between, and enrichment as operations that humans perform; however, there is no comparison operation.

Finally, a frequent cost-benefit analysis that participants performed while evaluating source patches seemed unique to variation foraging. Participants looked for source patches that were similar to the destination patch to minimize the cost of integration. This suggests that IFT models could benefit by including such analysis.

What makes a good variant?

Currently, the creation of variants in the environment is subjective; a programmer may or may not decide to save a particular variant of their code. The intermediate variants are important when programmers adopt exploratory programming, yet saving every single change would create too many variants for a person to use effectively. Understanding the points at which intermediate variants have the most significance can improve tool support for exploratory tasks.

Implications for Tools

Our results, although formative, have initial implications for tool design.

Organizing variants

Recall that participants' goals centered on understanding and comparing variants along different dimensions (e.g. game appearance, source code content, and chronology). The environment had limited support for these types of comparisons; participants managed to compare variants by opening only two files at a time, side-by-side, which was inefficient in the presence of hundreds of variants. Tools could support such comparisons between variants by highlighting similarities and differences along different dimensions. They could also support grouping and filtering variants along different dimensions. Finally, since variants were not linked, participants had to forage between variants by navigating out of a variant and into another by using the folder structure. Tools could ease navigation by linking variants through some of these dimensions.

Supporting effective navigation among variants

Tools to support variation exploration can improve user navigation. Such tools could include features such as:

1) Identify and highlight cues that reveal what makes variants different from or similar to one another.
2) Leverage automated techniques that analyze and summarize program changes to create descriptive change logs that highlight evolution history.
3) Extract major events and milestones in program evolution, and present a timeline of major landmarks in the history of the software.

CONCLUSION

This study is the first to investigate how novice programmers forage through past and present variants. Analyzing their foraging behavior through an information foraging theory lens produced the following insights:

- *RQ1 (types of information used):* Our results revealed new cue types signaling differences (e.g., update dates) and similarities (e.g., previous names) among variants; these cue types were extensively used by participants. We also found that participants reused cues to reduce the cost of searching for cues in the environment—a new finding that can inform tool development.
- *RQ1a (foraging between variants)*: Participants created stories about the evolution of the program, and then used these stories to guide their subsequent foraging between variants. Their foraging behavior fell into three distinct navigation patterns: unidirectional, bidirectional, and divide and conquer.
- *RQ1b (foraging within variants)*: Participants exploited structural similarities *between* source and destination variants' code structures to enhance their *within*-variant foraging in the source variant, by spotting patches within the source variant that would be easy to integrate into the destination. Differences between the source and destination patches acted as a negative scent, leading participants to abandon their within-variant foraging and return to between-variant foraging.
- *RQ2 (integration)*: Participants satisficed when integrating code between variants: they tried to find an appropriate patch that was similar to the destination patch, but if the cost of finding such a patch was too great, they modified some patch that seemed "good enough" or reimplemented the functionality from scratch.

Most important, this paper presents the first empirical evidence that IFT has gaps when applied as a theory of variation foraging. For example, IFT does not account for the temporal aspect of how participants developed stories, or how participants exploited structural similarities between different variants to locate patches of interest, or the significance to the participants of finding differences among similar variants. We believe that extending IFT to incorporate variation-specific constructs will enable us to extend and benefit from a theoretical foundation of how people forage in the presence of variants.

ACKNOWLEDGMENTS

We thank our participants for their help. This work was supported in part by NSF 1302113, 1314384, 1253786, 1314365, and 1439957 and David Piorkowski's IBM PhD fellowship.

REFERENCES

1. Joel Brandt, Philip J. Guo, Joel Lewenstein, Mira Dontcheva, and Scott R. Klemmer. 2009. Two studies of opportunistic programming: interleaving web foraging, learning, and writing code. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09), 1589-1598. http://doi.acm.org/10.1145/1518701.1518944
2. Margaret Burnett and Brad Myers. Future of End-User Software Engineering: Beyond the Silos. 2014. In International Conference on Software Engineering: Future of Software Engineering track (ICSE Companion Proceedings '14), 201-211. http://dx.doi.org/10.1145/2593882.2593896
3. Ed H. Chi, Peter Pirolli, and James Pitkow. The scent of a site: A system for analyzing and predicting information scent, usage, and usability of a web site. In proceedings of the SIGCHI conference on Human Factors in Computing Systems. ACM, 2000. http://dx.doi.org/10.1145/332040.332423
4. Ed H. Chi, Peter Pirolli, Kim Chen, and James Pitkow. Using information scent to model user information needs and actions and the Web. In Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 490-497. ACM, 2001. http://dx.doi.org/10.1145/365024.365325
5. Ed H. Chi, Adam Rosien, Gesara Supattanasiri, Amanda Williams, Christiaan Royer, Celia Chow, Erica Robles, Brinda Dalal, Julie Chen, and Steve Cousins. The Bloodhound project: Automating discovery of web usability issues using the InfoScent simulator. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '03). ACM, New York, NY, USA, 505-512. http://dx.doi.org/10.1145/642611.642699
6. Paul Clements and Linda Northrop. 2001. Software product lines: Patterns and practice. Addison-Wesley Professional.
7. Cloud9. 2015. Cloud9 – your development environment, in the cloud. Retrieved September 23, 2015 from https://c9.io/
8. Logan Engstrom, Garrett Finucane. 2015. Hextris. Retrieved September 23, 2015 from https://hextris.github.io/hextris/
9. Logan Engstrom, Garrett Finucane, Noah Moroze, Michael Yang. 2015. Hextris. Retrieved September 23, 2015 from https://github.com/Hextris/hextris
10. Scott D. Fleming, Chris Scaffidi, David Piorkowski, Margaret Burnett, Rachel Bellamy, Joseph Lawrance, and Irwin Kwan. 2013. An information foraging theory perspective on tools for debugging, refactoring, and reuse tasks. ACM Transactions on Software Engineering and Methodology 22, 2: 14 http://doi.acm.org/10.1145/2430545.2430551
11. Wai-Tat Fu, and Peter Pirolli. SNIF-ACT: A cognitive model of user navigation on the World Wide Web. Human–Computer Interaction 22.4 (2007): 355-412.
12. Björn Hartmann, Sean Follmer, Antonio Ricciardi, Timothy Cardenas, and Scott R. Klemmer. 2010. D.note: revising user interfaces through change tracking, annotations, and alternatives. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10), 493-502. http://doi.acm.org/10.1145/1753326.1753400
13. Björn Hartmann, Loren Yu, Abel Allison, Yeonsoo Yang, and Scott R. Klemmer. 2008. Design as exploration: creating interface alternatives through parallel authoring and runtime tuning. In Proceedings of the 21st annual ACM symposium on User interface software and technology (UIST '08), 91-100. http://doi.acm.org/10.1145/1449715.1449732
14. Reid Holmes and Robert J. Walker. 2013. Systematizing pragmatic software reuse. ACM Transactions on Software Engineering and Methodology 10, 2: 44. http://dx.doi.org/10.1145/2377656.2377657
15. Paul Jaccard. 1901. Étude comparative de la distribution florale dans une portion des Alpes et des Jura. Bulletin del la Société Vaudoise des Sciences Naturelles. 37: 547–579.
16. Amy K. Karlson, Greg Smith, and Bongshin Lee. 2011. Which version is this?: improving the desktop experience within a copy-aware computing ecosystem. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11), 2669-2678. http://doi.acm.org/10.1145/1978942.1979334
17. Scott R. Klemmer, Michael Thomsen, Ethan Phelps-Goodman, Robert Lee, and James A. Landay. 2002. Where do web sites come from?: capturing and interacting with design history. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '02), 1-8. http://doi.acm.org/10.1145/503376.503378
18. Ranjitha Kumar, Jerry O. Talton, Salman Ahmad, and Scott R. Klemmer. 2011. Bricolage: example-based retargeting for web design. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11), 2197-2206. http://doi.acm.org/10.1145/1978942.1979262
19. Sandeep K. Kuttal. 2014a. Leveraging variation management to enhance end users' programming experience. ETD collection for University of Nebraska - Lincoln.
20. Sandeep K Kuttal, A. Sarma, and Gregg Rothermel. 2013. Predator behavior in the wild web world of bugs: an information foraging theory perspective. In Proceedings of the IEEE Symposium on Visual Languages and

Human-Centric Computing (VL/HCC '13), 59-66. http://dx.doi.org/10.1109/VLHCC.2013.6645244

21. Sandeep K. Kuttal, Anita Sarma, and Gregg Rothermel. 2014. On the benefits of providing versioning support for end users: an empirical study. ACM Transactions on Software Engineering and Methodology 21, 2: 9. http://doi.acm.org/10.1145/2560016
22. Joseph Lawrance, Christopher Bogart, Margaret Burnett, Rachel Bellamy, and Kyle Rector. 2009. How people debug, revisited: an information foraging theory perspective. In Visual Languages and Human-Centric Computing (VL/HCC '11), 117-124.
23. Joseph Lawrance, Rachel Bellamy, and Margaret Burnett. 2007. Scents in programs: does information foraging theory apply to program maintenance?.In Visual Languages and Human-Centric Computing (VL/HCC '07), 15-22. http://dx.doi.org/10.1109/VLHCC.2007.25
24. Joseph Lawrance, Rachel Bellamy, Margaret Burnett, and Kyle Rector. 2008. Using information scent to model the dynamic foraging behavior of programmers in maintenance tasks. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08), 1323-1332. http://doi.acm.org/10.1145/1357054.1357261
25. Joseph Lawrance, Rachel Bellamy, Margaret Burnett, Kyle Rector. 2008. Can information foraging pick the fix? a field study. In Visual Languages and Human-Centric Computing (VL/HCC '08), 15-19 http://dx.doi.org/10.1109/VLHCC.2008.4639059
26. Joseph Lawrance, Margaret Burnett, Rachel Bellamy, Christopher Bogart, and Calvin Swart. 2010. Reactive information foraging for evolving goals. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10), 25-34. http://doi.acm.org/10.1145/1753326.1753332
27. Brad A. Myers, YoungSeok Yoon, Joel Brandt. 2013. Creativity Support in Authoring and Back-tracking. In Proc. Workshop on Evaluation Methods for Creativity Support Environments at CHI (CHI '13), 40-43.
28. Nan Niu, Anas Mahmoud, and Gary Bradshaw. 2011. Information foraging as a foundation for code navigation (NIER track). In Proceedings of the 33rd International Conference on Software Engineering (ICSE '11), 816-819.
29. Seymour Papert and Cynthia Solomon. 1989. Twenty things to do with a computer. In Soloway & Spohrer: Studying the Novice Programmer (SS '89), 3-27.
30. David J. Piorkowski, Scott D. Fleming, Irwin Kwan, Margaret M. Burnett, Christopher Scaffidi, Rachel K.E. Bellamy, and Joshua Jordahl. 2013. The whats and hows of programmers' foraging diets. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13), 3063-3072. http://doi.acm.org/10.1145/2470654.2466418
31. David Piorkowski, Scott Fleming, Christopher Scaffidi, Christopher Bogart, Margaret Burnett, Bonnie John, Rachel Bellamy, and Calvin Swart. 2012. Reactive information foraging: an empirical investigation of theory-based recommender systems for programmers. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12), 1471-1480. http://doi.acm.org/10.1145/2207676.2208608
32. David Piorkowski, Scott D. Fleming, Christopher Scaffidi, Liza John, Christopher Bogart, Bonnie E. John, Margaret Burnett, Rachel Bellamy. 2011. Modeling programmer navigation: A head-to-head empirical evaluation of predictive models. In Visual Languages and Human-Centric Computing (VL/HCC '11), 18-22. http://dx.doi.org/10.1109/VLHCC.2011.6070387
33. David Piorkowski, Scott D. Fleming, Christopher Scaffidi, Margaret Burnett, Irwin Kwan, Austin Z. Henley, Jamie Macbeth, Charles Hill, Amber Horvath. To Fix or to Learn? How Production Bias Affects Developers' Information Foraging during Debugging. In IEEE International Conference on Software Maintenance and Evolution (ICSME '15). http://dx.doi.org/10.1109/ICSM.2015.7332447
34. Peter Pirolli, Stuart Card. Information foraging in information access environments. In Proceedings of the ACM SIGCHI Conference on Human factors in computing systems (CHI'95), http://dx.doi.org/10.1145/223904.223911
35. Peter Pirolli, Wai-Tat Fu. SNIF-ACT: A model of information foraging on the World Wide Web. User modeling 2003. Springer Berlin Heidelberg, 2003. 45-54. http://dx.doi.org/10.1007/3-540-44963-9_8
36. Peter Pirolli. Rational analyses of information foraging on the web. Cognitive science 29.3 (2005): 343-373.
37. Peter Pirolli, Wai-tat Fu, Ed Chi, and Ayman Farahat. Information scent and web navigation: Theory, models and automated usability evaluation. In Proceedings of HCI International. 2005.
38. Peter Pirolli. 1997. Computational models of information scent-following in a very large browsable text collection. In Proceedings of the ACM SIGCHI Conference on Human factors in computing systems (CHI '97), 3-10. http://doi.acm.org/10.1145/258549.258558
39. Peter Pirolli. 2007. Information Foraging Theory: Adaptive Interaction with Information. Oxford University Press.
40. Mary Beth Rosson and John M. Carroll. 1996. The reuse of uses in smalltalk programming. ACM Transactions on Software Engineering and Methodology 3, 3: 219-253. http://doi.acm.org/10.1145/234526.234530

41. Chris Scaffidi, Chris Bogart, Margaret Burnett, Allen Cypher, Brad Myers, and Mary Shaw. 2009. Predicting reuse of end-user web macro scripts. In Visual Languages and Human-Centric Computing (VL/HCC '09), 93-100. http://dx.doi.org/10.1109/VLHCC.2009.5295290

42. Sruti Srinivasa Ragavan. Variations Foraging – Tasks. 2015. Retrieved January 08, 2016 from web.engr.oregonstate.edu/~srinivas/variations-foraging-tasks.html

43. Michael Terry and Elizabeth D. Mynatt. 2002. Side views: persistent, on-demand previews for open-ended tasks. In Proceedings of the 15th annual ACM symposium on User interface software and technology (UIST '02), 71-80. http://doi.acm.org/10.1145/571985.571996

44. Michael Terry, Elizabeth D. Mynatt, Kumiyo Nakakoji, and Yasuhiro Yamamoto. 2004. Variation in element and action: supporting simultaneous development of alternative solutions. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04), 711-718. http://doi.acm.org/10.1145/985692.985782

45. Young Seok Yoon and Brad A. Myers. 2014. A longitudinal study of programmers' backtracking. In Visual Languages and Human-Centric Computing (VL/HCC '14), 101-108. http://dx.doi.org/10.1109/VLHCC.2014.6883030