Reactive Information Foraging for Evolving Goals

Joseph Lawrance[1,2], Margaret Burnett[2], Rachel Bellamy[3], Christopher Bogart[2], Calvin Swart[3]

[1]MIT CSAIL
77 Massachusetts Avenue
Cambridge, MA 02139
lawrance@csail.mit.edu

[2]Oregon State University
School of EECS
Corvallis, Oregon 97331
{burnett,bogart}@eecs.oregonstate.edu

[3]IBM T.J. Watson Research
19 Skyline Drive
Hawthorne, New York 10532
{rachel,cals}@us.ibm.com

ABSTRACT
Information foraging models have predicted the navigation paths of people browsing the web and (more recently) of programmers while debugging, but these models do not explicitly model users' goals evolving over time. We present a new information foraging model called PFIS2 that does model information seeking with potentially evolving goals. We then evaluated variants of this model in a field study that analyzed programmers' daily navigations over a seven-month period. Our results were that PFIS2 predicted users' navigation remarkably well, even though the goals of navigation, and even the information landscape itself, were changing markedly during the pursuit of information.

Author Keywords
Information foraging theory, programming, field study

ACM Classification Keywords
D.2.5 [Software Engineering]: Testing and Debugging; H.1.2 [Information Systems]: User/Machine Systems—Human factors

General Terms
Experimentation, Human Factors, Theory

INTRODUCTION
When seeking information, how do people decide where to look? In the 90's, Pirolli and several colleagues first investigated this question by developing a new cognitive theory called information foraging [26]. Since then, studies of information foraging have focused on structured information-seeking tasks with well-defined goals on static, unchanging collections of documents. Yet, exploratory information-seeking tasks often have no clear goal defined in advance, although information encountered along the way often informs the goal. Furthermore, information seeking often involves context switches in a dynamic, changing information world. Can information foraging theory explain and model human behavior even given these kinds of challenges in day-to-day information-seeking tasks?

Information foraging theory is based on optimal foraging theory, a theory of how predators and prey behave in the wild. In the wild, predators sniff for prey, and follow scent to the place (patch) where the prey is likely to be. Applying these notions to the domain of information technology, the people in need of information (predators) sniff for information (prey), and follow information scent through cues in the environment to the places (patches) where the prey seems likely to be. Models of information foraging have successfully predicted which web pages human information foragers select on the web [9], and as a result, information foraging principles have become useful in designing web sites [22, 29]. Systems based on information foraging models have also streamlined information seeking time [8, 23], and helped evaluate web site design [10].

In this paper, we consider the fact that information foraging often occurs when the forager does not have a fully-formed goal, or when new information in the environment changes the forager's understanding of the goal. Although previous models could in principle change the goal (for example by explicitly adding keywords to a search query as with ScentTrails [23]), in those works the goal is an externally specified independent variable—a parameter to the model that does not change as the model runs. In contrast, in this paper we present a model in which the goal is an evolving *dependent* variable that changes depending on what the model observes over the course of a single model run.

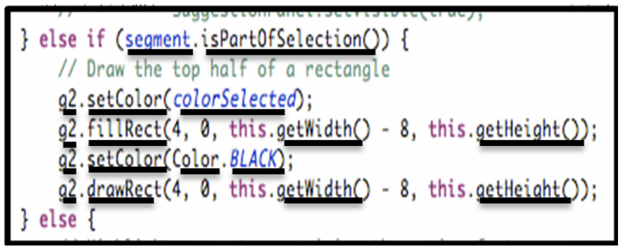
Programming is an ideal domain for studying these issues in information foraging theory for several reasons. First, previous work has suggested that debugging involves information foraging, from Brooks' early work on beacons [6] to prior debugging research that presented the PFIS (Programmer Flow by Information Scent) model of information foraging in debugging [18, 20]. Second, it challenges information foraging theory: debugging is rich in information seeking choices, where each screen offers *many* more navigation choices in a typical class than the reported average of 20 to 30 hyperlinks on a typical web page [4, 12]. For example, in Eclipse, every identifier links to another region of code (Figure 1). Third, programming exemplifies the situation in which the world changes due to frequent edits and updates as a team revises source code. This situation is increasingly important in the web as well, in web applications such as wikis and blogs. Fourth, since a bug can masquerade as something very different from its real cause, debugging epitomizes prey that may evolve.

This paper presents a new model of information foraging named PFIS2 (Programmer Flow by Information Scent 2). We investigate its ability to predict programmers' navigation during their day-to-day activities. This paper therefore makes the following contributions. First, the algorithm we developed is relatively simple to implement and affords flexibility in modeling human behavior, allowing experimentation with variations of the model. Second, our model of information foraging reacted to information edits during foraging activity. Third, the model reacted to incremental changes in programmers' conceptions of navigation goals and understanding of code during navigation. Finally, this paper presents a field study of professional programmers during seven months of their everyday work, which necessarily includes noise from phenomena such as interruptions, wild goose chases, and context switching.

BACKGROUND AND RELATED WORK

Information foraging theory was developed in the context of understanding web browsing [24, 25, 27]. Information foraging theory derives from rational analysis [1] the assumption that humans have adapted strategies that allow them to find information efficiently in the world around them, minimizing the cost to achieve their goals. The theory applies to information navigation in general, although application to any new domain requires defining the concepts relative to that domain.

The first model of information foraging theory was Pirolli and Card's ACT-IF [26], which models a user foraging for information using the Scatter Gather browser. It is a cognitive model, implemented in the ACT-R cognitive architecture [2]. This evolved into SNIF-ACT which models a user foraging on the web [7]. SNIF-ACT models scent as the relationships between linguistic concepts in a web user's mind. It represents these relationships via a spreading activation network initialized from on-line text corpora. The strength of a relationship between two concepts is assumed to be proportional to the frequency with which they are found near each other in the corpora. This kind of analysis has been shown to provide a good approximation to the way humans' mental associations work [17]. Given a goal (prey), the words in the prey activate nodes in the network. If nodes activated by the goal and words labeling a link



```
} else if (segment.isPartOfSelection()) {
    // Draw the top half of a rectangle
    g2.setColor(colorSelected);
    g2.fillRect(4, 0, this.getWidth() - 8, this.getHeight());
    g2.setColor(Color.BLACK);
    g2.drawRect(4, 0, this.getWidth() - 8, this.getHeight());
} else {
```

Figure 1. A few lines of source code. The underlines denote the navigational choices (links). This snippet alone consists of over 10 links, and represents a tiny portion of the class file.

have strong associations, they will become more active than words with weak associations. In this way, activation spreads through the network such that activation increases in nodes highly related to the original goal and decays if nodes are weakly or unrelated to the goal.

In SNIF-ACT, production rules encode users' possible actions such as Attend-to-Link, Click-Link, Backup-a-page, and Leave-Site. The model uses these rules to simulate a user's decisions about selecting a link versus leaving the site. For example, the model will normally select the link with the greatest scent; but if it determines that the scent of all the unvisited links is low enough, it will instead leave the current web site. In other words, if the scent is perceived to be stronger elsewhere, it is time to leave. SNIF-ACT 2 [13] adds Bayesian reasoning to decide incrementally when to leave a page from its experiences with the links and web pages visited from that page so far. The model will choose Backup-a-page when the perceived scent emanating from this page (calculated as a function of cues on the page looked at so far) is less than scent emanating from previously explored pages (calculated as a function of link scent on previously visited pages). In this way, it takes into account the impression of a page's overall remaining worth even if the user has not scanned the entire page. Our models are also incremental, but in a different sense: our models incrementally infer the information need over time.

The WUFIS (Web User Flow by Information Scent) algorithm [9, 10] also models scent and foraging, but does not require an underlying cognitive model. Instead it models many users navigating through a web site seeking information, predicting the probable number of users that would reach each page by following cues most related to a particular query. It does so by estimating the scent of a link on a web page as a function of the cosine similarity between words in the query and words in or near the link, represented as document vectors weighted by their term frequency inverse document frequency or TF-IDF [3]. Given the scent calculation of each link on each page, WUFIS creates a network representing the web site and weights each edge according to its proximal cue's scent. Then it uses spreading activation to predict where in the site most users, pursuing this query, would eventually end up. Whereas SNIF-ACT is a fully functioning cognitive model, and must be customized for each information foraging context being studied, WUFIS can be applied readily in new contexts. WUFIS can also work backward: its sibling algorithm, IUNIS (Inferring User Need by Information Scent) [9], takes the path a user traversed through a web site and infers what their original information need was.

Prior models of information foraging when debugging built on the WUFIS/IUNIS work. An executable model called PFIS (Programmer Flow by Information Scent) [18, 20] analyzed the topology of the source code, then calculated the scent of the cues that lead to each method, and finally propagated the scent along links with proximal cues containing scent relative to the prey, using spreading activation.

PFIS predicted navigation as effectively as the aggregate judgments of 12 human programmers, but to make these predictions, the model required descriptions of prey up front that defined the participants' tasks, and furthermore, PFIS did not account for changes to the source code. Therefore, PFIS was inadequate for modeling information foraging in which the prey and the source code evolved during information foraging.

Understanding the principles of information foraging has improved the design of web sites, and has resulted in systems that streamline information seeking [8, 23]; thus, we believe that debugging tools would also benefit from understanding information foraging in this domain. For example, despite efforts to develop source code search tools to help programmers navigate, Ko et al.'s empirical investigation of programmers' frequent information seeking during debugging showed that only half of the searches returned task-relevant code [16]. Tools that account for information scent in some way have produced impressive empirical results that suggest information foraging theory is useful in this domain (e.g., [11, 14, 28]). We aim to demonstrate quantitatively that information foraging models can predict people's real-world behavior in this domain, to support the theory as a foundation for tool development.

PFIS2: A NEW MODEL OF INFORMATION FORAGING

Constructs

Prior work adapted the theoretical constructs of information foraging theory to debugging [20]. We summarize those theoretical constructs, and also present PFIS2's *operational approximations* of those constructs.

- *Prey*: The knowledge the programmer seeks to understand how to fix the bug. In PFIS2: A bug description, e.g., a bug report.
- *Proximal cue*: A word, object, or behavior in the environment that suggests prey. Cues are signposts. Cues exist only in the environment. In PFIS2: Words in the source code, including comments.
- *Information scent*: Perceived "relatedness" of a cue to the prey (perceived likelihood of it leading toward prey). Scent exists only in the user's mind. In PFIS2: Inferred relatedness of a cue to the prey, as measured by amount of activation from a spreading activation algorithm.
- *Topology*: The collection of paths through relevant documents and on-screen displays through which the programmer can navigate. In PFIS2: An undirected graph through the source code, in which nodes/vertices are elements of the source code, and its edges are environment-supported links (defined below). Topology is environment-dependent.
- *Link*: A connection between two nodes in the topology that allows the programmer to traverse the connection efficiently. In PFIS2: "Efficiently" means at most one click or keystroke. The PFIS2 implementation works in Eclipse, which supports the links in Table 1. Because at most one click or keystroke is necessary to scroll between contiguous methods in the editor, PFIS2 recognizes links between contiguous methods.
- *Information patch*: A locality in source code, related documents or displays in which the prey might hide. In PFIS2: source code methods.

Element	Link type	Element
Workspace	Has	Projects
Project	Defines	Packages
Package	Defines	Class or Interface
Class or Interface	Imports	Class or Interface
Class or Interface	Extends	Class or Interface
Class	Implements	Interfaces
Class or Interface	Defines	Variables
Class or Interface	Defines	Methods
Variable	Has	Class or Interface
Method	Returns	Class or Interface
Method	Invokes	Methods
Method	Contiguous to	Methods
Method	Defines	Variables
Method	Uses	Variables

Table 1. PFIS2 recognizes these Eclipse-provided one-click (or less) links. "Contiguous to" represents the low cost of navigating to methods right next to each other in the text.

The PFIS2 algorithm

Because PFIS [18], an existing information foraging model of debugging, requires the description of prey up front to make predictions, and assumes that source code never changes, we realized that it was ill-suited to modeling the behavior of users during a field study where programmers' goals evolved and source-code changed. So we developed a new model called PFIS2.

The PFIS2 algorithm takes as input: the description of prey (if available), the patches the user has encountered in previous navigations, and the topological links available for navigation. (We gathered these inputs by instrumenting Eclipse using a plug-in we built for our field study.) In our field study, the PFIS2 algorithm updated whenever the user navigated or modified source code, but it can be run only whenever predictions are needed. Figure 2 summarizes the PFIS2 algorithm.

Step 1. A map of the world according to PFIS2

PFIS2 views the information world as the union of two graphs: the source code topology and the cues seen so far. PFIS2 scans for cues by looking at every word, separating by camel case, performing word stemming, and eliminating English stop words (relatively uninteresting words such as "the" and "and") and Java's reserved words. Each cue becomes a vertex. The source code topology provides additional vertices: one for each package, class, interface, method, and variable. The edges are the topological links (Table 1) plus the edges that connect cues to the location(s) in which the cue occurred.

```
Initialize:
  construct empty Graph G, Activation vector A, path
  read description of prey (e.g., bug report)
  insert each word in description into G and A, with A[word]=1

Update predictions whenever the user opens source code
files not seen before, changes source code, or navigates:
  read changes to source code seen/changed
  read user's newest navigations, currentMethod

  Step 1 Update G's source code seen by the user so far: its
  cues, locations, and edges between them.
  Step 2 Activate each method in navigations by recency:
    append navigations to end of path
    x := 1.0
    for i=length of path down to 1:
      A[path[i]] := x
      x := x * 0.9
  Step 3 Spread activation through A for k iterations, decay-
  ing at the rate of $\alpha$=0.85:
    toSpread := [currentMethod]
    for k iterations:
      for each node in toSpread:
        w := 1 / (number of neighbors of node in G)
        for each neighbor of node in G:
          A[neighbor] := A[neighbor] + (A[node]*w*$\alpha$)
      toSpread = all neighbors processed in this iteration
  Return updated vector A of activation scores.
```

Figure 2. The PFIS2 algorithm. (Variants in Table 2)

More formally, PFIS2's graph $G = (V_1 \cup V_2, E_1 \cup E_2)$. G is an undirected graph. V_1 is the set of all source code locations, and E_1 is the set of links among them (by the operational definition of links from the previous section). V_2 is $C \cup V_1$, where C is the set of all cues (words) in source code elements, and each edge in E_2 connects an element $c \in C$ to an element $v_1 \in V_1$ if c occurs in v_1.

Step 1 maintains this graph. For example, suppose the user has opened the RSSOwl package in Eclipse, and in that package the user opened the RSS class shown in Figure 3. The prey is the following bug report, in its entirety:

"Archive feed items by age."

Step 1 creates nodes and edges reflecting the code in Figure 3: an edge from package RSSOwl to class RSS, from class RSS to each of its methods and variables, and from RSS's methods to each of the classes, methods, and variables that they use. For each edge, it creates appropriate vertices if they are not already in G. A portion of the graph produced so far is shown in Figure 4's left.

Step 1 also connects cue vertices for each word in the bug report and each word near links to the locations in which they occur. For example, the cue "create" connects to methods `createArchiveDir`, `createWorkingDir`, and `startupProcess`, as shown in Figure 4's right, because that word appears in each of those methods, as shown in Figure 3. (In PFIS2, "near" means "in the method".) Spreading activation will eventually push those cues' effects into other methods, as we explain shortly.

Note that PFIS2 knows more about code the user has seen than it does about code not yet seen. In the example, PFIS2 knows which methods in the RSS class are contiguous to which other methods, all the methods they call, and all the words shared among the methods it has seen so far. In contrast, the user has never looked at the File class, so PFIS2 knows only the name (and words in the name) of the `File` class and `exists` method, which it knows because a new `File` instance was declared and the `exists` method was called by methods in the RSS class. There are many other methods in the RSSOwl package about which PFIS2 knows nothing at all at this point.

Step 2: Setting up the activations

Activation values are how PFIS2 tracks which nodes in G are most active. Thus, having updated graph G in Step 1, in Step 2, PFIS2 updates the activation vector A for the loca-

```
...
public static void main(String[] args) {
  startupProcess();
  new RSS();
}
private static void createWorkingDir() {
  String homePath = '/.rssowl';
  File homeDir = new File(homePath);
  if (!homeDir.exists()) {homeDir.mkdir();}
  GlobalSettings.WORK_DIR = homePath;
}
private static void createArchiveDir() {
  File archive = '/archive';
  if (!archive.exists()){archive.mkdir();}
}
private static void startupProcess() {
  createWorkingDir();
  createArchiveDir();
}
```

Figure 3. Example RSSOwl source code.

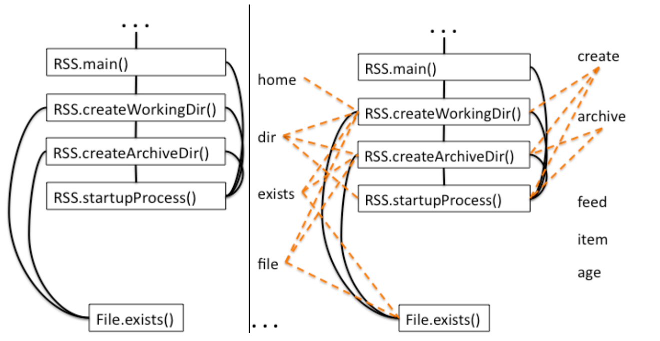


Figure 4. (Left) The topology of example RSSOwl source code (G_1=(V_1, E_1)).

(Right) The union of the source code topology with encountered cues $G = (V_1 \cup V_2, E_1 \cup E_2)$. Nodes in strongly connected subgraphs of cues and methods reinforce each other when they become highly active. For example, createWorkingDir() and createArchiveDir() share 6 adjacent nodes and an edge, so activation of either node also activates the other node.

tion nodes that have become "active" recently. Suppose the user navigated from `main` to `startupProcess` to `createArchiveDir` to `createWorkingDir`. Step 2 would result in the following nodes having these activation values, due to the exponential decay in path distance from the current position: `createWorkingDir` 1, `createArchiveDir` 0.9, `startupProcess` 0.81, `main` 0.729. If `createWorkingDir` occurs twice in *path*, only the most recent activation is considered.

In the initialization step, PFIS2 already activated words that were in the bug report: *archive* 1, *feed* 1, *item* 1, *age* 1. Based on which words the user follows up persistently in navigations, the activation level of these cues and methods in which they appear will increase or decrease in Step 3.

Step 3: Spreading activation

In Step 3, PFIS2 spreads activation along edges to model the user's evolving interests. In its first iteration, it spreads activations from the current method by increasing the activation of the cues and locations linked to the current method in the topology. For example, since the current method is `createArchiveDir`, activation will spread directly to class RSS, to any classes used in `createArchiveDir` (in this case `File`), to the methods it calls (`exists`, `mkdir`), to the methods contiguous to it (`createWorkingDir`, `startupProcess`), and to cues in it (create, archive, dir, file, exist, …). At this point, it has reinforced one cue from the bug report, namely archive, because the user has navigated to methods related to that cue.

At this point, all direct neighbors (location nodes and word nodes depicted in Figure 4's right side) of the current method in G have been activated. Thus, PFIS2 thinks each has scent relative to the user's current interests. In the next iteration, all of those vertices spread further decayed scent to *their* neighboring locations and cues (one of which is the current method), and in the third iteration, every vertex that just received activation spreads a further decayed scent to their neighbors. (It has been our experience that the activation ranks become stable after three iterations.) PFIS2 then returns a ranked list of activation scores for methods, which is PFIS2's estimation of each method's scent. The highest ranked method is PFIS2's prediction of the next method to which the user will navigate.

Note that the edges constrain where activation can flow: methods near to or called by active methods become more active, and cues in active methods become stronger. Thus, whenever the user navigates, even if it is to a method PFIS2 did not expect, that method becomes highly activated, alerting the model to the user's evolving interests. Activation then spreads to cues in that method, incrementally modeling the evolution as to *which* cues will next be of highest interest next.. The more active cues, in turn, strengthen method nodes containing those words, especially those with direct links to the active methods. This combination of cues and history in spreading activation allows PFIS2 to quickly evolve its notion of the prey.

Representation of cues as nodes with undirected links to their origins is a practical alternative to other information foraging approaches that estimate locations' relevance to cues. This approach is akin to TF-IDF (used in PFIS and WUFIS); both track degree of word sharing among documents. Other approaches have used latent semantic analysis [17] or point-wise mutual information [21], which require corpora, leading to the need to find a suitable corpus. The cues-as-vertices approach of PFIS2 does not need a corpus. Instead, PFIS2 conceptually uses the past code it has seen as its "corpus" for the purpose of informing semantic relevance. For example, if the programmer pursues a cue not literally found in the bug report but discovered in the code as they navigate, activation of that cue will spread to all locations containing that cue. Thus, by continually informing relevance estimations with new navigations, we sacrifice some predictive power at the start of a navigation path in exchange for greater flexibility in modeling evolving discoveries affecting the user's definition of the prey.

FIELD STUDY METHODOLOGY

To evaluate PFIS2 and our research contributions, we conducted a field study. We collected logs of participants' day-to-day work in Eclipse over seven months, using a plug-in that we built. We first tested the plug-in on a few pilot participants. After we addressed performance issues identified by the pilots, we invited a group of professional programmers at IBM to install the plug-in and participate by engaging in their normal, day-to-day activities.

Two professional programmers used our plug-in for the full seven months of the field study. Those two participants (one male, one female) worked at IBM and used Eclipse in their day-to-day work. Both were team leaders for components of Rational products, including a visualization toolkit, a server product, and Jazz (built on Eclipse). Beginning in the summer of 2008 and ending in the winter of 2009, we captured participants' every action within Eclipse, a total of 2.3 million actions, generating 4,795 predictions of between-method navigations. Note, it is these 4,795 predictions that are the experimental units used in our analysis, not the outcomes of participants' actions themselves.

Experimental variants of PFIS2

We experimented with different versions of PFIS2 to compare their predictions by manipulating the initial activation vector (step 2 of the algorithm) as shown in Table 2.

PFIS′ ("PFIS-prime") is our baseline system for comparison. PFIS′ is conceptually the same as the original PFIS [18]. Whereas PFIS computed information scent as cosine similarity weighted by TF-IDF, PFIS′ computed information scent as the activation of cues in the spreading activation graph. We implemented PFIS′ using PFIS2 because PFIS did not handle enough details of navigation to allow a fair comparison of the "explicit prey only" base situation. Like PFIS, PFIS′ does not inform prey with cues observed along the way.

In all versions, the model predicted the *next navigation step* given where the programmer had just been.

Variant	Step 2 activation strategy
PFIS′ (baseline)	None. (*path*=empty list.)
PFIS2-ImmedScent-Explicit	Activate current method only. (*path*=[*currentMethod*].)
PFIS2-AllScent-Explicit	Activate all methods encountered so far with most recent weighted most heavily. (the *path* variable is as given in Figure 2.)

Table 2. These variants of PFIS2 were constructed by changing the activation in step 2 of the algorithm.

Quality measure

To assess and compare the above model variants, for each navigation step, we ranked current predictions according to strength (activation value), removing the prediction of the user's current method. Then we checked to see what rank was held by the prediction that matched the participant's actual navigation. For example, if a participant navigated to method Y, we looked up Y in the model's list of predictions. Suppose the predictions were: X first, Y second, and Z third. Since the participant navigated to the second ranked method, the model's prediction rank would be 2. For each model variant, we statistically evaluated these prediction ranks at each of the users' navigations. Note that this measures a very strict criterion, requiring that PFIS2 predict where participants went in their *very next* click.

RESULTS

Our plug-in recorded Eclipse *session*s, defined as the time between when a participant started Eclipse until he or she terminated Eclipse. We analyzed the 76 sessions that involved active navigation and edits, reinitializing after each session. The median session lasted 94 minutes, but many sessions lasted several hours and even days (the mean was about 28 hours). The data represented over 2000 hours of observation of 4,795 between-method navigations. We used the data to evaluate our information foraging models.

Incrementally redefining the prey in a changing world

Past work on information foraging has viewed prey as well defined up front. We hypothesized that PFIS2 would provide a more accurate model of users' prey when debugging than an unchanging representation of the prey.

For the original description of the prey, we expected to use bug reports, but although our participants had told us they do use issue tracking, they did not do so during the seven months of our study. However, they retrospectively provided us design documents and time-stamped revision logs describing the bug fixes, such as in Figure 5. The revision logs, like bug report titles, could uniquely identify source code artifacts and changes that closed bugs [15, 19]. (Bug reports have widely differing quality [5], which has implications for their use in models, a point we return to later.) Our evaluation was done by replaying logs against the model variants; thus we used the descriptions like bug reports, namely as explicit prey at the beginning of each relevant session, as determined by manually tying them by participant and timestamp to the relevant sessions. These sessions covered 1,209 of our participants' between-method navigations.

```
Cleaner version of the output format.
ROLLBACK & other optimization.
Classpath, PPTx with names merged, name changes in
formatter.
```

Figure 5. Three of the revision logs used as explicit descriptions of the prey.

Comparing these models revealed that informing prey with scent encountered during foraging more accurately modeled user behavior than an unchanging representation of the prey. PFIS′, which used an unchanging notion of prey, was the worst model of the variants in Table 2; its median prediction rank was 99 (out of a median of over 700 possible navigation choices). On the other hand, PFIS2-ImmedScent-Explicit, which incrementally updates its notion of the prey using only the most recent navigation, had a median rank of 31 (out of 700+ choices). PFIS2-AllScent-Explicit was much more effective than either PFIS2-ImmedScent-Explicit or PFIS′. PFIS2-AllScent-Explicit, which starts with a bug report and updates the prey in light of cues encountered at each step along the way, predicted participants' navigations with a median rank of 8 (out of 700+ choices). This means that about 604 of the participants' 1,209 navigations were in PFIS2-AllScent-Explicit's top 8 choices. In fact, 10% of the places participants navigated were this model's first choice. Figure 6 shows the distributions of prediction ranks for these three models.

The differences among these three models were statistically significant[1]: PFIS2-AllScent-Explicit was a significantly better model than PFIS2-ImmediateScent-Explict (paired Wilcoxon signed rank test, N=1209, Z=-24.131, p<.001), which in turn was better than PFIS′ (paired Wilcoxon signed rank test, N=1209, Z=-20.377, p<.001). These differences clearly show the importance of the acquisition of incremental scent throughout the user's journey.

In Figure 6, note the long tails. The end of the long tail of the best model, PFIS2-AllScent-Explicit, was rank 658 as its worst prediction. In total, it had 88 predictions at rank 100 or worse (cropped off in the figure), but this long tail covered only about 7% of all the navigations.

A session in detail

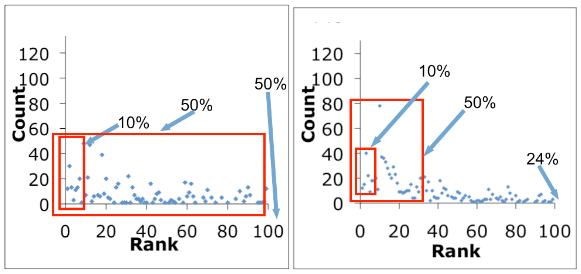
A look at individual sessions helps to clarify the strengths and weaknesses of the best model: PFIS2-AllScent-Explicit. Figure 7 shows one session in which the participant navigated between methods 22 times before ending the

[1] Figure 6 shows that the distribution of prediction ranks for each model was not normal, hence the nonparametric tests.

session. Patterns like this occurred frequently in almost all of the sessions. In the example shown, its first prediction was rank 9. The quality of the first prediction should be strongly affected by the quality of the bug report itself.

The predictions tended to improve over time, but at some point, the user would do something wildly unexpected: navigate between seemingly unrelated methods, a "surprise" from PFIS2's perspective. The second navigation in Figure 7 was a surprise (prediction rank 109). Recall that about 7% of the navigations were surprises to PFIS2-AllScent-Explicit. We believe such surprises are inevitable in real-world situations such as refactoring or switching contexts.

Notice in Figure 7 how quickly PFIS2 corrected its course when such unexpected navigations arose. It almost always recovered within one or two navigations, getting back to the single-digit rankings very soon (median recovery time: 1 navigation). Out of 1209 predictions, the worst it did was a run of three in a row in 3-digit territory. (Compare this to its best run: 24 in a row in single-digit territory.) In Figure 7, it took two navigations for it to recover, at which point its rank improved to 2 (fourth navigation).



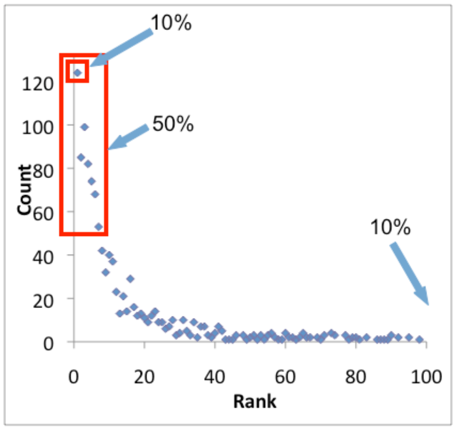


Figure 6. Histograms of the prediction rank distributions by model. The x-axis represents the rank of the prediction (only predictions ranked 1-100), the y-axis represents the frequency of those ranks. Callouts denote the percentage of data (e.g., 50% of PFIS′ prediction ranks were greater than 100). Top left: PFIS′. Top right: PFIS2-ImmedScent-Explicit. Bottom: PFIS2-AllScent-Explicit.

In contrast, PFIS′ had more surprises; in fact, 50% of the navigations produced three-digit rankings (median 99). For PFIS′, the only "expected" methods were those with scent related to the explicit prey or close to methods with scent related to the explicit prey. Figure 8, which shows PFIS′ performance for the same session as Figure 7, demonstrates where course correction could have helped PFIS′. For example, the user went back to the same method as in the original surprise two more times, and also visited two methods with closely related functionally to that one, all of which were high-rank surprises to PFIS′.

What about the influence of time? For PFIS2-AllScent-Explicit, predictions after surprises usually got continually better until the next big surprise, as in the running average graph in Figure 7 right. However, as Figure 9 demonstrates, length of the *session* did not help; perhaps because the user's prey changed substantially enough that early knowledge may not have been useful many navigations later. Nor did sheer number of sessions help, because we did not use a machine learning approach to try to learn about a participant or project; rather, each session was predicted independently of any previous session's data.

In general, then, our characterization of PFIS2-AllScent-Explicit, which took into account cues encountered all along the user's journey, is as follows:

- The model often started out reasonably well (depending on the bug report), and got better with the consideration of cues observed along the way.

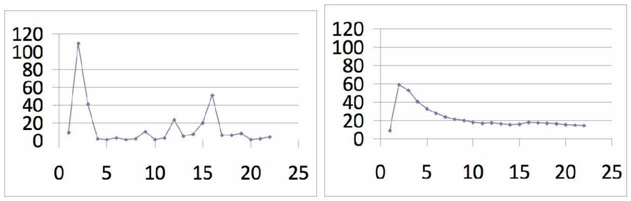


Figure 7. Left: One session predicted by PFIS2-AllScent-Explicit. X-axis: time as navigation actions. Y-axis: the rank of the prediction. Patterns like to this were common among sessions. Right: Running average rank for the same session.

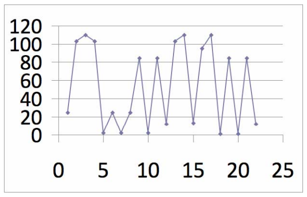


Figure 8. PFIS′ predictions for the same session as Figure 7.

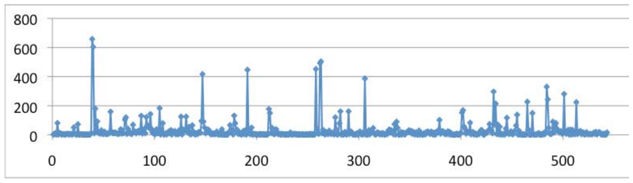


Figure 9. A session of more than 500 navigations. PFIS2-AllScent-Explicit's accuracy was not tied to session length.

- The user changed direction, surprising the model.
- The model rapidly recovered (within one click at least half the time), and usually had a continual improvement trend (as in Figure 7 right) until the next substantial change in direction.

What if there are no bug reports?

Sometimes, no explicit descriptions of prey are available, such as when an information seeking task is described only verbally. Lack of an explicit prey description is common in web foraging, and also occurs in debugging. For example, the participants worked for seven months on tasks that never found their way into actual bug reports.

Is it reasonable to suggest that information foraging models can predict people's behavior when the model has no explicit description of the prey? The precedent of IUNIS [9] having some success at predicting people's initial goals from their navigations supports this possibility, as does our PFIS2-AllScent-Explicit model's gainful incorporation of the cues the user observed along the way. To investigate this question, we used the PFIS2 algorithm as in the previous section but without providing any explicit prey at all. In terms of Figure 2, there was no *description*; the algorithm omits the following step: insert each *word* in *description* into *G* and *A,* with *A*[*word*]=1

Surprisingly, the results were *better* than with explicit prey descriptions. PFIS2-AllScent turned in a remarkable median rank of 3 (out of 700+ choices): almost 2,400 of the participants' 4,795 total navigations were in PFIS2's top 3 choices. In fact, in this model, 27% of the methods participants navigated to were the model's number 1 choice. Figure 10 shows the results of running these models on our full set of 4,795 between-method navigations.

Table 3 shows that the improvement over the best model of the previous section was highly significant. The less ambitious version, PFIS2-ImmedScent, also performed significantly better than its counterpart from the previous section, PFIS2-ImmedScent-Explicit, but not as well as the previous section's best, PFIS2-AllScent-Explicit, as shown in Table 3. Comparing all models using solely the 1,209 navigations for which all could be evaluated, the differences between the models were all statistically significant (see Table 3).

Interestingly, PFIS2-AllScent did not have unique strengths and weaknesses. Instead, it had mostly the same patterns as PFIS2-AllScent-Explicit, but tended to do just a little better on everything. Both had similar patterns of surprises, often in the same places. PFIS2-AllScent's advantage was pervasive: 89% of its individual predictions were at least as good as PFIS2-AllScent-Explicit's. Its within-session medians and means were as good or better on 100% of the sessions, as were the ranks of its first-prediction-of-sessions 88% of the time and of its session-end predictions 75% of the time.

PFIS2-AllScent's superiority over PFIS2-AllScent-Explicit might seem to suggest that a less-than-ideal bug report, such as the ones we used, is worse than having none at all. This is indeed one explanation. Recall that our participants wrote many of the descriptions as post-navigation revision logs. Thus, our results could mean that those descriptions did not capture the participants' initial verbal prey descriptions well; perhaps the log descriptions were indeed worse than bug reports. This interpretation is a testament to the power of cues-observed-along-the-way, providing very good performance in PFIS2-AllScent-Explicit despite poor initial descriptions.

An alternative explanation is also a testament to the importance of cues observed along the way. When bug reports provide incomplete information, people *must* rely more on cues along the way than on the report, simply because they have little alternative. We posit that PFIS2-AllScent's remarkable performance is precisely because it captures the importance of cues observed along the journey.

Under either explanation, the success brought about by incremental scent in *both* the explicit-prey and non-explicit-

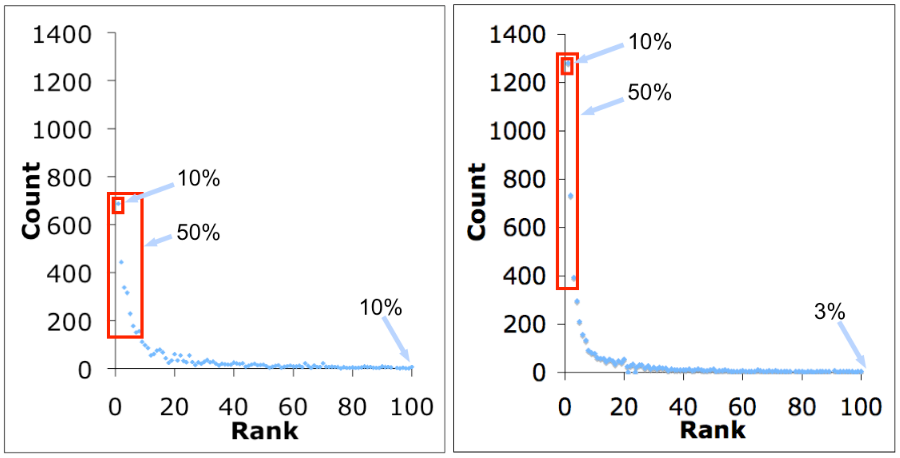


Figure 10. (Left) PFIS2-ImmedScent (Median rank: 8). (Right): PFIS2-AllScent (Median rank: 3). Callouts denote the percentage of data.

prey models is consistent with the fact that people start navigating and, along the way, they learn information that influences where they will go next. People rarely assemble a complete goals list up front and then pursue it to the bitter end. Rather, cues encountered along the way build a better picture of what a sensible goal may be. PFIS2's ability to model incremental redefinition of prey captures this aspect of human behavior.

DISCUSSION

The scale of our data set contributes to the validity of the results, which were based on 4,795 navigation actions from over 2000 hours of observation of participants' day-to-day work over seven months. Even so, the observations came from just two participants who may have peculiar navigation habits that influenced PFIS2's predictions. Also, although we expect the approach to generalize to other foraging activities that have incrementally evolving goals, this premise must be evaluated empirically.

From a theory perspective, PFIS2 differs from previous models in that it adjusts its notion of the prey itself after each navigation step. In essence, PFIS2 explicitly models a forager's evolving goals in response to new information encountered during navigation. Detailed analysis of the timeline of our model's predictions showed how the model corrected its notion of the prey based on cues users encountered along the way.

From an algorithm perspective, PFIS2 has the advantage of its inherent simplicity. Unlike information foraging models such as SNIF-ACT, PFIS2 does not require a cognitive modeling engine. It also does not require a corpus in order to estimate semantic relatedness. Rather, it operationalizes the theory using only the users' overt actions and the information environment, without trying to model fine-grained details of what goes on in the user's mind. In short, it is a rational analysis of navigation. Its simplicity may increase the viability of incorporating information foraging models directly into tools.

As for PFIS2's contributions to the programming domain, PFIS2's modeling results demonstrate the strong foraging orientation of our participants' seven months of programming. This suggests that PFIS2, while *not* a programming tool, can inform the design of programming tools and source-code libraries. The theoretical constructs suggest questions designers of code should ask themselves such as: Does this code have sufficient information scent with respect to its functionality? By showing where programmers are likely to navigate, model-informed tools could evaluate the design of code with respect to how well it facilitates programmer navigation. Incorporating the model into issue trackers and IDEs may also help ease program navigation.

Predictions	Model variant	Wilcoxon Z, p<.001
Worst	PFIS′	
Poor	PFIS2-ImmedScent-Explicit	} Z= -24.131
Good	PFIS2-ImmedScent	} Z= -19.221
Better	PFIS2-AllScent-Explicit	} Z= -20.377
Best	PFIS2-AllScent	} Z= -21.501

Table 3. The paired Wilcoxon signed rank test (N=1209) compared each model with the model in the row above it. Each model was significantly better than the model above it (p<.001). Because explicit prey were available for only 1209 of the predictions, N=1209 for all comparisons.

Our research raises many open questions regarding the approach per se, the best ways to operationalize it, and its application in the programming domain. For example, how much would a high-quality initial description of the prey (e.g., a high-quality bug report) influence the predictions of each model? What types of cues most influence programmers' navigation? Would better accounting for enrichment activities, such as the arrangement of windows and tabs, and note-taking, improve the model? Finally, would upfront provision of synonyms and linguistic relationships improve on PFIS2's graph-based treatment of cues, in which words' relatedness must be "discovered" over time?

CONCLUSION

In this paper we presented PFIS2, an information foraging model of navigation during everyday software development tasks. PFIS2 tackles three real-world situations. First, its perspective of prey changes incrementally, modeling users' changing notions of the prey. Second, PFIS2 operates under a very high number of navigation choices. Third, the information world can change significantly while foraging.

We evaluated PFIS2's suitability for modeling real-world foraging by logging 4795 navigations by two professional programmers over seven months and comparing how well PFIS2 could predict where these programmers really navigated. Our empirical results revealed that:

- PFIS2 accurately predicted our participants' navigation. The most successful PFIS2 variant (PFIS2-AllScent) achieved a median prediction rank of 3, and even predicted our participants' navigations as its first choice 27% of the time.
- PFIS2 predicted our participants' navigations successfully even in the absence of explicit descriptions of the prey such as bug reports.
- PFIS2's success was tied to course correction. Its incremental notions of prey allowed it to recover from big surprises very quickly (median: only one navigation for the two best variants).

Incremental prey is a novel contribution to information foraging theory. Its inclusion in the model demonstrates how foragers respond to changes in the world as they occur. It also demonstrates how opinions of the "right" scents to pursue changes incrementally in response to cues encountered along the way.

Most important, the study's results demonstrate the external validity of an information foraging model of human navigation behavior, even in the face of changing information and goals that evolve over time.

ACKNOWLEDGMENTS
This work was supported in part by AFOSR FA9550-09-1-0213, by NSF ITR-0325273, by an IBM International Faculty Award, and by J. Lawrance's IBM PhD Scholarship.

REFERENCES
1. Anderson, J. R. *The Adaptive Character Of Thought*. Erlbaum, Hillsdale, NJ, USA, 1990.
2. Anderson, J. R. *Rules of the Mind*. Erlbaum, Hillsdale, NJ, USA, 1993.
3. Baeza-Yates, R. and Ribeiro-Neto, B. *Modern Information Retrieval*. Addison Wesley Longman, Reading, MA, 1999.
4. Becchetti, L., Castillo, C., Donato, D., Baeza-Yates, R. and Leonardi, S. Link analysis for web spam detection. *ACM Transactions on the Web 2*, 1 (2008).
5. Bettenburg, N., Just, S., Schroeter, A., Weiss, C., Premraj, R., and Zimmermann, T., What makes a good bug report? In *Proc. FSE 2008*, ACM Press (2008), 308-318.
6. Brooks, R., Towards a theory of the comprehension of computer programs. *Int. J. Man-Mach. Stud. 18*, (1983), 543–554.
7. Card, S., Pirolli, P., Van Der Wege, M., Morrison, J., Reeder, R., Schraedley, P., & Boshart, J. Information scent as a driver of web behavior graphs: Results of a protocol analysis method for web usability, In *Proc. CHI 2001,* ACM Press (2001), 498-505.
8. Chi, E., Hong, L., Heiser, J., Card, S. ScentIndex: Conceptually reorganizing subject indexes for reading. In *Proc. IEEE VAST*, IEEE (2006), 159-166.
9. Chi, E., Pirolli, P, Chen, K. and Pitkow, J, Using information scent to model user information needs and actions on the web. In *Proc. CHI 2001*, ACM Press (2001), 490-497.
10. Chi, E., Rosien, A., Supattanasiri, G., Williams, A., Royer, C., Chow, C., Robles, E., Dalal, B., Chen, J., Cousins, S. The Bloodhound project: Automating discovery of web usability issues using the InfoScent simulator, In *Proc. CHI 2003*, ACM Press (2003), 505-512.
11. Cubranic, D., Murphy, G., Singer, J. and Booth, K. Hipikat: A project memory for software development, *IEEE Trans. Soft. Engr. 31*, 6 (June 2005), 446-465.
12. Fetterly, D., Manasse, M., Najork, M. and Wiener, J. A large-scale study of the evolution of web pages. In *Proc. 12th International World Wide Web Conf.*, ACM Press (May 2003), 669-678.
13. Fu, W. and Pirolli, P. SNIF-ACT: A cognitive model of user navigation on the world wide web, *Human-Computer Interaction 22*, 4 (Nov. 2007), 355-412.
14. Hill, E., Pollock, L., and Vijay-Shanker, K. Automatically capturing source code context of NL-queries for software maintenance and reuse, In *Proc. ICSE 2009,* IEEE Computer Society (2009), 232-242.
15. Ko, A., Myers, B., and Chau, D. A linguistic analysis of how people describe software problems, In *Proc. VLHCC*, IEEE (2006), 127-136.
16. Ko, A., Myers, B., Coblenz, M., and Aung, H., An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks, *IEEE Trans. Software Engineering 32*, 12 (Dec. 2006), 971-987.
17. Landauer, T. K. and Dumais, S. T. A solution to Plato's problem: the Latent Semantic Analysis theory of acquisition, induction and representation of knowledge. *Psychological Review, 104, 2* (1997), 211-240.
18. Lawrance, J., Bellamy, R., Burnett, M., and Rector, K. Using information scent to model the dynamic foraging behavior of programmers in maintenance tasks, In *Proc. CHI 2008*, ACM Press (2008), 1323-1332.
19. Lawrance, J., Bellamy, R., Burnett, M. and Rector, K. Can information foraging pick the fix? A field study, In *Proc. IEEE VLHCC*, IEEE (2008), 57-64.
20. Lawrance, J., Bogart, C., Burnett, M., Bellamy, R., and Rector, K. How people debug, revisited: An information foraging theory perspective. *IBM Technical Report RC24783* (2009). (Under review.)
21. Manning, C. and Schutze, *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA, May 1999.
22. Nielsen, J. Information foraging: Why Google makes people leave your site faster http://www.useit.com/alertbox/20030630.html. June 30, 2003.
23. Olston, C., Chi, E. ScentTrails: Integrating browsing and searching on the web. *ACM Trans. Computer-Human Interaction*, 10, 3 (2003), 177-197.
24. Pirolli, P. and Card, S., Information foraging in information access environments. In *Proc. CHI 1995,* ACM Press (1995), 51-58.
25. Pirolli, P. Computational models of information scent-following in a very large browsable text collection. In *Proc. CHI 1997*, ACM Press (1997), 3-10.
26. Pirolli, P. and Card, S. Information foraging, *Psychology Review* 106, 4 (1999), 643-675.
27. Pirolli, P. *Information Foraging Theory: Adaptive Interaction with Information*. Oxford University Press, New York, NY, USA, 2007.
28. Robillard, M. P. and Murphy, G. C. Concern Graphs: Finding and describing concerns using structural program dependencies, In *Proc. ICSE 2002*, ACM Press (2002), 406-416.
29. Spool, J., Profetti, C., and Britain, D., Designing for the scent of information, *User Interface Engineering*, (2004).