Version Control Systems: An Information Foraging Perspective

Sruti Srinivasa Ragavan, Mihai Codoban, David Piorkowski, Danny Dig, Margaret Burnett

Abstract—Version Control Systems (VCS) are an important source of information for developers. This calls for a principled understanding of developers' information seeking in VCS—both for improving existing tools and for understanding requirements for new tools. Our prior work investigated *empirically* how and why developers seek information in VCS: in this paper, we complement and enrich our prior findings by reanalyzing the data via a theory's lens. Using the lens of Information Foraging Theory (IFT), we present new insights not revealed by the prior empirical work.

First, while looking for specific information, participants' foraging behaviors were consistent with other foraging situations in SE; therefore, prior research on IFT-based SE tool design can be leveraged for VCS. Second, in change awareness foraging, participants consumed similar diets, but in subtly different ways than in other situations; this calls for further investigations into change awareness foraging. Third, while committing changes, participants attempted to enable future foragers, but the competing needs of different foraging situations led to tensions that participants failed to balance: this opens up a new avenue for research at the intersection of IFT and SE, namely, creating forageable information. Finally, the results of using an IFT lens on these data provides some evidence as to IFT's scoping and utility for the version control domain.

Index Terms—Human factors in software desgin, Software engineering, Version control

1 Introduction

SOFTWARE ENGINEERING (SE) is an information-intensive activity. Empirical studies have revealed that, as part of their day-to-day software engineering activities, developers ask several questions from *"why is this code implemented this way?"* to *"what are other team members working on?"* [27], [28], [52]. To answer such questions, developers seek information from various sources, such as the web, bug repositories, documentation or even other team members. One such information source is the project's version control system (e.g., Git, Subversion, Mercurial) [28].

Version Control Systems (VCS) are a rich source of information: they contain the entire development history of a project. SE researchers routinely leverage this rich information to multiple ends, such as to predict bugs [50], merge conflicts [6] and to recommend APIs to programmers [36], to name a few. However, even though studies (e.g., [28]) reveal that developers do seek information in VCS, surprisingly little research focuses on the details: what information they seek, how and why. As a result, as SE researchers, we are limited in our understanding of developers' activities and, as tool builders, we do not understand how well existing VCS tools support developers' needs and where new tool opportunities lie.

To address this gap, our prior empirical work [12] characterized the motivations, strategies and barriers to developers' information seeking in VCS, lifting the results to a three-lens model for software history. In this paper, we complement our prior empirical work and overlay a new perspective—that of a theory.

We re-analyze empirical data from our prior study [12] from the perspective of Information Foraging Theory (IFT) to gain fresh insights into *how* developers seek information in VCS and how VCS tools can better aid developers.

1.1 Why Theory?

Our end goal is to better support developers' information seeking. For that, we need to build good tools that will solve the right problems. Empirical studies, such as our prior work [12] are great for revealing phenomena (including problems), but they provide limited justifications for *why* the phenomena/problems happen. Theories, on the other hand, provide these justifications. This explanatory power of theories can help tool builders identify the right problems—not just the symptoms—that need to be solved. Further, because theories can tell *why* something happens, they can also reason in the opposite direction to predict what will happen in a given situation [22]. This predictive power can be leveraged for engineering good tools: for example, tool builders can theoretically reason about how a user will use a tool, or evaluate early on whether and why a tool will work (or not)—even before actually implementing the tool [22].

- *Sruti Srinivasa Ragavan is with the School of Electrical Engineering and Computer Science at Oregon State University, Corvallis, OR 97331. E-mail: srinivas@eecs.oregonstate.edu.*
- *Mihai Codoban is with the Tools for Software Engineers group at Microsoft, Redmond, WA 98052. Email: micodoba@microsoft.com.*
- *David Piorkowski is with IBM Research AI at IBM Thomas J Watson Research Center, Yorktown Heights, NY 10598. Email: david.piorkowski@ibm.com*
- *Danny Dig is with the School of Electrical Engineering and Computer Science at Oregon State University, Corvallis, OR 97331. E-mail: digd@eecs.oregonstate.edu.*
- *Margaret Burnett is with the School of Electrical Engineering and Computer Science at Oregon State University, Corvallis, OR 97331. E-mail: burnett@eecs.oregonstate.edu.*

1.2 Why IFT?

Information Foraging Theory (IFT) exemplifies the explanatory and predictive powers a good theory brings. IFT has successfully *explained* people's information seeking behaviors in various domains (including SE [29], [30], [32], [37]) and its *predictive* powers have guided the practical design and evaluation of various tools and information environments, such as websites and web search engines [10], [11], information visualizations [8] and SE tools [10], [39]. These prior successes of IFT in diverse domains encouraged our choice of IFT for the VCS domain.

We also chose IFT because of its generalizability. Successful solutions from one domain can be abstracted into generic IFT vocabulary as design principles or patterns (e.g., web design guidelines [53], principles for IDE navigations [21], [41], SE tool design patterns [15], [34]) to be reused in other domains. For this reuse, the phenomena in the "target" domain have to be framed in an IFT's vocabulary: that is what this study achieves. We frame version control activities in an IFT vocabulary so as to reuse existing IFT design patterns, without reinventing the wheel.

1.3 Does IFT apply to VCS?

Prior research has not applied IFT to the VCS domain and so one of the key goals of this paper is to evaluate the scope of validity of IFT in the VCS domain. By applying IFT to VCS, we can evaluate how well IFT is able to explain and predict phenomena related to version control foraging, and thereby, we can gather evidence as to whether IFT's scope extends to version control activities (or not).

1.4 Contributions

The key contributions of this paper are:

- IFT-based reanalysis of data from [12] to gain fresh insights into how developers seek information in VCS,
- evidence of the utility and appropriateness of applying IFT to the domain of version control.
- IFT-informed design solutions to address the barriers to developers' information seeking in VCS,
- discussion of new avenues for IFT research motivated by version control activities.

2 Background

Information Foraging Theory (IFT) [44] is a theory of how people seek information. Rooted in evolutionary psychology, IFT assumes that human information-seeking abilities have evolved similar to animals' food-foraging strategies. Therefore, to explain information seeking, Pirolli and Card turned to animals' food foraging behaviors, eventually deriving IFT from the optimal food foraging theories [54].

2.1 Constructs

IFT uses a small set of constructs (summarized in Table 1). *Predator* is the person (e.g., developer) hunting for information or *prey* (e.g., bug location). Just like animals hunt for their prey in foraging grounds, in IFT, a predator forages for prey in the *information environment* (e.g., VCS).

According to IFT, the information in the environment occurs in *patches* (e.g., commits, branches, files) and the patches contain *information features* (e.g., words, colors). The prey is actually a set of information features contained in one or more patches. A predator can go from one patch to another via a *link* (e.g., scrolling to go from one commit to another, clicking on commit message open a commit).

Central to how predators forage for prey are the notions of *cues* and *scent*. Cues are information features associated with links: they tell the predator what information might be found at the other end of the link (e.g., text on a hyperlink, icons, file names, commit messages). Just as animals sniff at various cues (e.g., hoofprints, fur bits) and take the path with the strongest scent of prey, IFT posits that human predators will also attend to use the *scent* from the cues to guide them to their prey.

2.2 Cost-value proposition

More formally, according to IFT, a predator treats the information foraging problem as an optimization problem, trying to get to the prey in the most efficient manner.

In most information-rich environments (e.g., VCS), a predator has several available foraging choices (e.g., several possible patches s/he can go to, several links s/he can follow), each bringing different informational value and at different cost (i.e., time, no. of clicks, cognitive costs). IFT says that, among these choices, a predator will choose the option that will maximize the rate of gain of valuable information (or the value-to-cost ratio).

In reality, however, the predator often does not know the actual value or cost associated with an action (e.g., if a predator has never visited a commit, s/he does not know what value it exactly has). Therefore, to make foraging decisions, the predator guesses the values and costs based on *cues* available in the environment. This value-cost estimation is what constitutes *scent*, i.e.:

$$Predator's\ choice = \text{Max(scent)} = Max\left[\frac{\text{Expected value}}{Expected\ cost}\right]$$

This equation, called IFT's cost-value proposition, is the mechanism by which IFT makes predictions about how a predator will forage.

As we will see in this paper, we can use this cost-value proposition to: 1) understand how developers will forage, 2) evaluate and design tools and 3) gather evidence to assess the scope of IFT's validity in VCS domain.

TABLE 1: CONSTRUCTS OF IFT

Construct	Definition	Example in VCS
Predator	Person foraging for information	Software developer
Prey	Information that the predator is seeking	When was this bug introduced?
Information environment	Environment where the foraging happens	Version control system
Patch	Locations in the environment, contains information features.	Commit, list of commits
Link	Connection between patches	Clicking on (link) commit message opens the commit.
Cues	Provide hints about what information might be at the other end of the link.	Words in commit messages, timestamp
Scent	Predator's estimate of value / cost ratio (in their heads)	Similar words = higher scent

3 Related Work

3.1 Information Foraging Theory

Pirolli and colleagues derived IFT to explain information seeking in large document collections [42] and information visualizations [8]. They then applied IFT extensively to the web domain, to explain and predict people's web browsing behaviors [10] and to evaluate websites [9], [11], eventually laying the foundations for web usability [53].

In the software engineering (SE) domain, Ko et al. first suggested IFT as a theory for developers' information seeking [27]. Ever since, researchers have applied IFT to explain information-seeking in various SE tasks, such as requirements engineering [37], debugging [31], [32] and program maintenance [40]. Recent work has also investigated programmers' foraging among program variants during exploratory programming [45], [46].

Although the majority of IFT work in SE is on explaining developer behavior via empirical studies or computational models, researchers have also leveraged IFT for SE tool building: Piorkowski et al. adopted PFIS2, an IFT computational model, to recommend methods to programmers [39], Henley et al. leveraged IFT to design IDE navigation affordances [20] and Perez et al. used IFT to guide the design of Pangolin, a program comprehension tool [38].

Researchers have also gone beyond specific tools and lifted IFT's insights to generic and reusable design principles and patterns. Notable examples include a community-curated IFT design patterns catalog for SE tools [16, 34], principles for IDE navigation design [20] and Piorkowski et al.'s four fundamental ways of improving tool design [41]. These patterns and principles are framed in generic IFT vocabulary and are transferable to other domains.

This paper builds on prior IFT research, particularly the design patterns catalog [34], to understand developers' foraging in VCS and to improve VCS tools. Along the way, this paper also evaluates the scope of IFT in the version control domain (where IFT has not been applied before).

3.2 Information seeking in VCS

SE practitioners have used VCS tools for over three decades, but few studies have investigated how developers forage in version control history. Our prior study [12], on which this paper is built, is one of the first studies to do so. However, researchers have investigated and aimed to support specific version control activities, which we summarize in the rest of this section.

3.2.1 Change awareness

Empirical studies have looked into developers' collaborative needs in SE, including change awareness. Notable papers in this area include Guzzi et al.'s study of developers' collaboration practices [18], Gutwin et al.'s categorization of general vs. specialized change awareness needs [16] and DeSouza et al.'s characterization of developers' forward and backward impact management needs [14].

Based on these empirical studies, researchers have also built tools to meet developers' collaboration needs. Some tools, such as ProjectWatcher [17] and Bellevue [18], support developers' general change awareness, while others, such as FastDash [4], Cassandra [23] and Palantir [48], specifically focus on conflict detection and awareness.

3.2.2 Locating specific commits

Studies have revealed that developers ask several questions about code (e.g., why is some code the way it is), (e.g., [52], [28]), but research has largely overlooked questions such as: what kinds questions do developers ask in VCS, how do they find answers to those questions or how to support them. A notable exception is [55], in which Tao et al. discuss how developers understand the changes in a commit and how tools can aid this activity. Our prior study [12] also began investigating these activities.

3.2.3 Committing changes

In the realm of committing changes, some studies have attempted to understand the nature of commits developers create. For example, Alali et al. investigated the characteristics of a typical commit [1], Brindescu et al. [5] investigated the commit sizes made by Git and SVN users, Kawrykow and Robillard studied commits to understand the nature of changes made on a project [24] and Kirinuki et al. investigated tangling of changes in commits [26].

Other researchers have used commit characteristics, such as commit sizes [33], commit message content [25] or pull request information [56] to predict the nature of changes in the commit(s).

Even others have attempted to help developers in the commit process, such as by automatically generating commit messages [13]. Yet, these studies do not look into how developers create commits, and why.

In summary, existing literature is largely focused on specific version control activities, in contrast, we treat version control activities as a whole. We also differ from existing work in that we bring to version control activities a theory: as a result, we emphasize on *whys*: *why* developers need the information they are looking for, *why* they go about looking for it the way they do, *why* they face challenges and *why* existing tools meet (or not) developers' needs. Answering these why questions are important for tools builders, who might otherwise make incomplete or inaccurate assumptions about developers and their needs.

4 Methodology

As mentioned earlier, this paper presents the results from an IFT-based *re-analysis* of the data from a prior study [12]. Therefore, we first discuss the methodology of the original study and then discuss the reanalysis procedure.

4.1 Prior study: Interviews

We recruited 14 developers from across 11 companies via convenience sampling. On an average, participants had ~13 years of professional software development experience. They used diverse VCS (Git, SVN, TFS, Bazaar) and clients (e.g., command line, Github, Stash).

We used semi-structured interviews: we started with a fixed set of anchoring questions (listed in the study's website [57]), but followed interesting tangents based on participants' responses [51]. Each interview lasted 40-90 minutes. Each participant received $50 in compensation.

For analysis, we used qualitative methods. Since ours was the first study on the subject, we did not have a prior codeset. So, we used open coding to build our codeset, following Campbell et al.'s advice [7] for segmentation, codebook evolution and inter-rater agreement issues.

First, we transcribed the interviews. We then segmented the transcripts topically: each time the interviewer asked a new question, or participant changed the topic of the conversation, we created a new segment. Since segmentation required subjective interpretation and contextualization of the participants' responses [7], the second author, who also conducted the interviews, performed the segmentation.

After all interviews were transcribed and segmented, the first and second authors coded the transcripts proceeding one at a time. During each coding session, the two coders *independently* coded each segment, allowing multiple codes per segment. Since a prior code set was not available, the coders coded participants' motivations, strategies and barriers using their own code names and schemes.

After independently coding a transcript, the coders compared their code sets. First, they resolved disagreements in code names: for example, one coder used a code name "*why is this this way*", while the other coder used the name "change rationale"; eventually, the coders agreed on the name "why is this *this* way" and updated both the code set and the coded transcripts. After such renaming, the coders received an inter-rater agreement (IRR) of 65% (Jaccard index) averaged across all sessions. These IRR levels are consistent with the measures Campbell et al. report [7].

The coders then resolved any disagreements in code assignment via "negotiated agreement", discussing the code assignment until they reached agreement [7]. Along the way, they added new codes, removed or merged existed codes and disambiguated code descriptions. For such codebook changes, the coders recoded previously-coded interviews to reflect the updated code book.

With such a process, the codebook changed at the end of each coding session. The final codebook from each session became the starting codebook for the next session. Consistent with prior studies involving semi-structured interviews [7], it took 10 interviews for the codebook to stabilize. The final codebook is provided in [57].

At the end of all coding sessions, the coders achieved an IRR of 97.4%. The coders then grouped the codes into larger emerging themes.

4.2 Prior study: Survey

Interviews provided rich data about VCS usage, but only from a small sample size. To validate and quantify our interview findings with a broader demographic, we designed a survey based on our interview findings. The survey consisted of multiple-choice questions (except for one open-ended question on commit messages). For each multiple-choice question, we included an "other" field to capture any new data that did not surface in the interviews. The survey questions are available on [57].

We advertised the survey via Reddit and Twitter. Of the 217 survey repondents, 80% had >5 years of software development experience and 84% were from the industry.

As mentioned earlier, the survey contained an open-ended question on what a commit message should contain. We analysed the responses to this question qualitatively, via open coding. The first and second authors coded the data and, via negotiations, reached 93.5% agreement. This codeset is also found in the accompanying website [57].

4.3 Re-analysis procedure

For the reanalysis, we mapped high-level codes (code categories) from the prior study [12] to an IFT vocabulary. Specifically, (1) *motivation* codes from [12] were about why developers seek information in VCS; in IFT, these reasons are called *foraging goals*. (2) *Strategy* codes were about how participants foraged from the perspectives of costs and values, patches, cues and/or diet and (3) *enabler* codes were what helped them do so; we mapped both these codes to *foraging strategy* codes that participants adopted and/or that tools supported. (4) *Barrier* codes captured participants' difficulties and (5) *wishes* codes were what participants wanted (but didn't have), so we mapped both to *foraging barriers*. The above list is the entire list of high-level codes from the prior codeset, and the low-level codes did not change. Since coding in [12] was done on the low-level codes, no recoding was necessary. The entire codeset and code mappings are available in the study's website [57].

This codeset mapping resulted in coded data instances that correspond to IFT (e.g., patches, diet, foraging goals). Given this, the rest of the re-analysis was to use the instances in the data of IFT concepts predicted using the theory's predictions, and to analyze whether and how these predictions were realized in the data.

5 Results 1. Change Awareness Foraging

Participants engaged in three main foraging activities: 1) *foraging for change awareness*, 2) *foraging for specific information* and 3) creating commits *with future foraging in mind*. This section discusses the first of these, and Sections 6 and 7 discuss the other two.

5.1 Change awareness foraging: a diet problem

Change awareness—staying up-to-date with the latest changes on a project—has previously been established to be important to developers (e.g., [12], [16], [28]), who tend to use version control systems to achieve it [12]. For our participants, their change awareness efforts amounted to solving what IFT terms the "*diet problem*" [43].

The IFT diet problem is a predator deciding what prey, among all available preys, he/she should consume as part of his/her "information diet" to get the maximum value per cost [43]. Applied to change awareness, the IFT diet selection problem is a developer selecting the changes to gain awareness about, within the time s/he want to spend on it.

IFT's diet models predict for optimal diet selection: a predator will choose to consume high value prey and exclude low value prey, even if the latter might be abundant and/or cheap to consume. This pattern has indeed been shown in other software engineering settings [40], and in our participants' foraging also it appeared. As Fig. 1 shows, participants did not waste time consuming all available changes; they selected only the changes most relevant to them to be part of their information diet.

P12: "I do not read every single commit … If something does not look as [if] it is needed, I shall ignore it."

Of course, what was relevant varied for each *particular* developer in each *particular* situation.

> *Insight #1*: Participants' dietary choices were consistent with traditional IFT diet models. This provides evidence that IFT might apply to change awareness foraging, and allows prior IFT results on diets to be leveraged for change awareness tools.

Implication for tools: Since the selection of diet in change awareness matched well to other IFT work on diet selection, tool builders can leverage IFT-oriented design solutions to support diet selection, such as the IFT-based design patterns compiled by Nabi et al. [34]. These design patterns crystallize decades of prior IFT research, abstracting designs in multiple successful SE tools so that they can be reused in tools for other similar situations, including for change awareness diet selection.

For example, the above compilation provides patterns such as specification matcher, notifier, dashboard, structural relatedness, recommendations and impact location that tool builders can leverage to help developers locate interesting (or high value) changes. There is already some evidence that doing so is appropriate for change awareness: several existing change awareness tools, such as GitHub's watch, Palantir's notifications and FastDash's dashboard, that are empirically shown to be successful, already implement some combination of these design patterns.

5.2 Change awareness's foraging: differences from many other SE foraging situations

Change awareness foraging also differed in nuanced ways from traditional notions of foraging.

Difference #1: The prey is easy to find. In traditional foraging, the prey is often elusive. For example, a developer might forage through a large codebase, seeking prey from multiple places in the code, using a combination of cues and experience to find relevant locations, with many deadends likely along the way (e.g., [42]). In contrast, in VCS, the prey is simply "what changed?", which most VCS environments present to developers with a single user action (e.g., via a pull action).

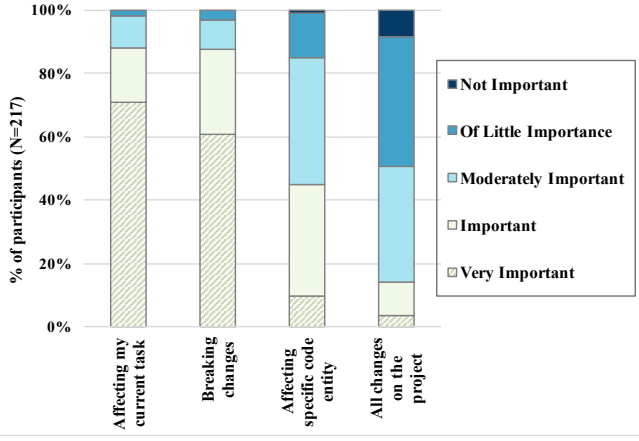

Fig. 1. Change awareness diet. Participants' change awareness diets were highly selective (i.e., they did not want to learn about all changes) and personalized (e.g., changes affecting *my* task).

Difference #2: Foraging to ease future foraging costs. Since the prey is so easy to find, why do developers need to forage at all in VCS tasks? The answer for our participants was that, in their change awareness foraging, they foraged to ease *future* foraging costs, mostly in the following two ways. They either foraged the changes to understand which ones will and will not require them to forage for its details at a *future* time (next, or much later); or, they foraged to avoid additional costs in future foraging situations (e.g., avoid merge conflicts, which can impose extensive foraging costs to resolve). For example, P4 foraged to avoid future merge conflicts:

P4: "If I … *need to be more cautious*, then I'll do a TkDiff <on> the files that are most crucial, see what changes have been made since I last looked." (emphasis added.)

Difference #3: Consume prey now, or save it for later? The third difference was in when predators *consumed* the prey. In traditional notions of foraging, a predator finds *and* consumes the prey within a task context; e.g., when debugging, a developer finds the bug location(s) and works with the found code to fix the bug. In contrast, during change awareness, participants used the found prey mostly to make timing decisions on when to consume it, based on the value/cost of now vs. later. For example, sometimes they consumed the prey immediately, other times they decided it could be consumed as part of a later diet (which they might actually consume later or not). P12 explains these value/cost decisions on the change awareness prey which arrived via his emails:

P12: "I go through email typically twice a day... [But] I have a couple of folders of email that, if I get any email, I'll look at it fairly quickly, within like 30 minutes or so… those are changes that are introduced to an *important repo* and *I want to know fairly quickly if something happened*". (emphasis added).

Difference #4: Lightweight. The final difference was in *how* participants allocated their time. In other SE research on foraging, the predator may be required to stubbornly pursue a specific prey often at high costs, as the needed prey may only exist in a specific patch (e.g., exact location of a bug). However, in change awareness situations, participants often had to become aware of multiple high-value changes. As a result, they adopted a lightweight approach: for each relevant change, they mostly spent only low costs, to gain only incomplete or "partially thorough" (P1) understanding of changes.

P6: "[I] look at the history *really quick*, see what has happened."

P9: "... each commit that you care about sends you an email, just *scan* the subject lines … you can *kind of see* the direction that the code base is going." [emphasis added in both quotes].

> *Insight #2:* Change awareness foraging showed several differences from most SE foraging:
> 1) the prey is easy to find,
> 2) much of the foraging is to ease future foraging,
> 3) actually prey consumption might be immediate or later, and
> 4) foragers adopted a lightweight approach.

Tool design implication: One view of the low-cost lightweight strategy supported so well by VCS tools is as a

model of good design for other foraging tools. Version control tools allow developers to easily maximize their value-to-cost ratio by enabling them to forage for high value information across multiple patches at very low cost. Specifically, the cost of gaining high-value information in a patch (e.g., reading commit message summary) as well as the cost of going from one patch to another (e.g., by scrolling through a list) are both low to enable this foraging. Such low costs in the environment can be viewed as an examplar for other SE tools (e.g., for debugging, IDEs) to strive for in supporting similar foraging needs in other SE tasks.

5.3 An unfulfilled foraging need: bulk change awareness

A foraging problem participants' version control tools did not address was their need for bulk change awareness. Participants' need for change awareness "in bulk" arose when they had too many changes to catch up with (e.g., after a vacation, or in large open source projects). In these cases, even though participants spent even less than usual time foraging within each commit, their overall foraging costs were high because of the high volume of commits.

The situation was so burdensome that some participants came up with explicit foraging strategies just to deal with it. Some participants kept up with commits and emails even during vacations:

> P3: "Look at emails to keep track… while on vacation."

Others who caught up with changes after the vacation, skimmed pull requests and release tags (groups of commits) which were far lesser in number than commits.

> P12: "I found that Github pull requests are one of the most helpful ways to do that… I'll look at all the pull requests that have been opened or changed since I was out…I'll go quickly through the merged and closed ones just to see what happened, then save the still opened ones for later..."

Even preferred to meet with other team members (adding to the team member's cost) to catch up with changes instead of foraging through all that data:

> P6: "...If I am just getting back and there has been a lot going on, I will usually prefer to have a *one-to-one meeting* with *various individuals* that are involved ..." (Emphasis added.)

Implication for tools: Our results suggest that tools for change awareness should facilitate deferred prey consumption (e.g., implement the cart pattern for developers collect to changes for future vetting), help developers estimate the future value of change awareness (e.g., based on an open issue or upcoming feature) and aid bulk change awareness. Existing tools have largely overlooked these aspects, leaving gaps for new change awareness tools to fill.

6. Results 2: Foraging for Specific Information

6.1 Foraging Traditionally for Specific Information

When looking for *specific* information, our participants' foraging needs (Table 2), are consistent with questions developers asked in other foraging studies in SE [31], [40], [46] (e.g., where is the bug located?, what does this code do?, what are the dependencies?, where shoud I reuse code from?). Thus, in seeking answers to those questions, our participants engaged in foraging behaviors consistent with those in prior studies. For example, consider P3's foraging behaviors during a debugging task:

> P3: "[I] …looked at commits for the last couple of days, looked at a particular solution, read the messages, read the diffs, talked to people."

Here, the foraging activities P3 mentions fall into traditional foraging categories:

- *enrichment*, to modify the environment for convenient foraging (e.g., filtering),
- *between-patch foraging*, to choose a profitable patch to forage in next (e.g., read commit message to choose which commit to go to), and
- *within-patch foraging*, to assimilate the information within a patch (e.g., read diffs to grasp a commit).

This similarity with traditional notions of foraging suggests that, to support foraging for specific information, VCS tool builders can harvest prior IFT tool design patterns or improvements. To demonstrate the feasibility and utility of such transfer of results, we employ Nabi et al.'s design patterns catalog [34] to address existing problems in VCS (listed as barriers in Table 3).

> Insight #3: While foraging for specific information, participants' foraging goals and activities were consistent other kinds of foraging in SE, suggesting that IFT might apply to these kinds of version control activities.

We place our suggestions under the four fundamental ways in which, according to IFT, tools can support foragers in their traditional foraging activities [41]:

1) helping developers accurately estimate costs,
2) helping developers accurately estimate values,
3) reducing actual foraging costs, and
4) increasing actual value in the environment.

6.2 IFT-Informed VCS design: supporting developers' estimates of cost and value

In IFT, a predator's ability to forage efficiently depends on how well he/she can estimate costs and values for various actions. In our study also, accurate estimates allowed participants to make informed and efficient foraging choices, whereas inaccurate estimates sometimes caused them to make poor foraging choices, leading to disappointment.

TABLE 2. FORAGING FOR SPECIFIC INFORMATION (TRADITIONAL FORAGING BEHAVIORS)

Foraging goal	Definition	Where participants foraged
Selectively compose changes	Find specific commits (e.g., bug, feature) to cherry-pick into other branches.	Commits pertaining to a specific bug or feature to that need to be cherry-picked.
Change impact analysis	Find which areas of code are impacted by a change, what tests need to be run.	Commits relevant to the specific changes (e.g., other commits modifying same code or tests).
Debug	Find when a bug was introduced, how the code at that time was.	Bug-introducing commit.
Understand code / change rationale	Understand why a code snippet was implemented a certain way.	Commit where the code was added or modified.

P4 faced such disappointment when he could not estimate costs while foraging for when a certain line of code came into being (Table 3: traceability to history). At first he navigated to the oldest version of that line, expecting that to be all his foraging costs. But once there, P4 realized the line had been moved from elsewhere, and he had to trace the history of the file the code had been moved from.

> P4: "…that line came into being in such and such a version but the reason it came into being is that a chunk of code got moved from here to there. So then you need to look at the previous version to see when that line came into being and that will refer you back farther … what CVS tells you is not actually true."

At this stage, P4 was disappointed because: 1) he could not find the prey in the location he had expected to find it, 2) the foraging costs were higher than what he had expected, and 3) he had no idea how much *more* it would cost him to get to his prey—at best, it might be just a few clicks away, but in the worst case, he might have to click through many, many code moves "without any end in sight" [41].

Similar disappointments also surfaced as part of participants' value estimations, when the actual value was much lower than what the participants expected. For example, P9 encountered a commit that contained mostly white-space changes, and few actual code changes. But he had no way of guessing that beforehand, without actually going through all the modifications within that commit.

> P9: "… we have our code style, so the tool reformats the code for you… you can have a 100 changed lines and only one is an actual code change [the rest are white-space changes]."

Implications for tools: These traditional foraging needs provide tool builders opportunities to harvest IFT design patterns that help developers with value and cost estimation [34]. Some of these patterns, such as cue decoration, signpost and visualizations, are already present in existing tools, in the form of commit messages, change summaries on getting the latest version and branch visualizations. However, as the barriers in Table 3 reveal, gaps remain (e.g., due to white space and redundant changes or unidentified code moves and renames) for which tools could better support developers' estimation needs.

6.3 IFT-Informed VCS design: improving actual foraging costs and values

Participants also faced difficulties because the actual costs of foraging in the environment were too high, or there was too little valuable information. In fact, as we shall see next, most of Table 3's barriers were due to poor *actuals*.

Information overload: low actual value. VCS history often contained too many commits (and, sometimes, commits contained too many changes). This excess of information led participants to experience information overload.

The challenge with information overload is that the bulk of the available information might not be relevant to a predator's foraging, resulting in low information value for the foraging. For example:

> P11: "…there could be some noise from commits I don't care about. Sometimes it is hard to filter out changes … for the merging … if they did a big refactor so they renamed a bunch of fields, they are not the person I need to talk to … ."

To improve the actual information value, Nabi et al.'s catalog [34] recommends capabilities for *filtering* out less valuable changes. Although most VCS tools provide some filtering capabilities—including branches and tags that offer implicit filtering—it appears from our results that they might be insufficient and/or hard to find and use (P5), calling for improvements.

High foraging costs in understanding commits: Even with valuable information, foraging was sometimes hard due to high within-patch and between-patch costs. High *within patch costs* were mainly due to tangled changes, limited cues (e.g., line added vs. line moved) and the lack of grouping and filtering capabilities *within* commits (e.g., group all changes for a method rename). High *between-patch costs* were because important information needed to understand commits were often scattered across other patches (e.g., other commits, branches) and even other environments (e.g., bug reports, emails, documents); developers had to navigate between these patches to obtain their prey.

To bring down both between-patch and within-patch costs, IFT's design patterns can help. For between-patch costs, tools such as Hipikat, CodeBroker and GitHub implement *gather together* design pattern. Combined with

TABLE 3. BARRIERS: IFT-BASED INTERPRETATIONS AND SOLUTIONS

Barrier	Meaning	Interpretation as one of four fundamental improvements in IFT[40]	Example solutions from Nabi et al.'s design pattern catalog [16]	% of participants (N=217)
Non-informative commit messages	Lack of details in commit messages about the changes in a commit.	Hard to estimate value in a commit	Problems arise in committing changes, as discussed in Section 7.	66
Tangled changes	Multiple, not necessarily coherent, changes tangled in the same commit.	High within-patch costs in understanding a commit		54
Information overload	Too much, and often, more irrelevant than relevant, information.	Low value, due to too many irrelevant changes	*Filtering* to weed out irrelevant information	47
Traceability to versions	Fragmented history of a line due to code move, file moves or renames.	High between patch costs going between the fragments	*Gather together* related fragments	32
Interpreting diffs	Limited cues, lack of filtering within commits, making them hard to understand.	High within-patch costs while understanding a commit	Feature decoration to highlight intents, move vs. add changes	32
Tool limitations	Missing tool features (e.g., filter, group, visualize).	[Discussed as part of other barriers.]		20
Traceability to requirement	Fragmented information makes it hard to relate a change to requirements (or vice versa)	High between patch costs going between commits and requirements	*Gather together* related fragments	20
Traceability to architecture	Limited support to view a change in the context of the entire project / its dependencies	High between patch costs going between commits and architectural details	Visualization showing changes vs. entire project	17

other patterns such as *structural relatedness*, *impact location* and *lexical similarity*, these tools gather together disparate, but related artifacts (e.g., bug reports, issues, related commits) to be recommended to developers. For within-patch costs, most VCS tools decorate the information features to allow easy processing: for example, in most VCS, changed code is decorated with surrounding code and code modifications are decorated in red and green. Similarly, HistoRef [19] implements *feature tracing* to decorate changes with similar intent with colors and numbers and ChangeDistiller [24] groups within-commit changes to eliminate "non-essential" changes (e.g., all method occurences affected by a method rename).

> *Insight #4:* VCS tools offer several opportunities for improvements, both in terms of cost/value estimates as well as in terms of actual costs and values. Nabi et al.'s catalog [34] contain design patterns that achieve each of these improvements.

Implications for SE research: However, as other researchers [47], [35] have pointed out, state-of-the-art VCS tools suffer from fundamental limitations: (1) they only capture coarse grained changes and not fine grained changes as developers edit code, (2) the changes are primary text-based and not AST-based (e.g., as in [19]). These, in turn, limit our ability to implement some of the existing patterns, such as reduce duplicate information (e.g., replace multiple instances of a method rename as one refactoring change). Our results, from an IFT's perspective, reiterate the need for overcoming these limitations to better support developers' foraging activities.

Finally, as a word of caution in interpreting this section, our intent in this section is *not* completeness. In fact, one of the limitations in this section is that we do not explore all possible interpretations of the barriers, consider all possible solutions among IFT design patterns, or include other information environments (e.g., emails, bug reports) participants foraged in. *Instead, the main point of this section is the formative evidence that IFT might apply to VCS foraging* and that tool builders could profit from existing IFT research for building and improving VCS tools.

7. Creating Patches for Future Foraging

7.1 Committing changes: will future foraging be easy?

In VCS, developers not only consume, but also produce the information that others, or themselves, might forage in at a later time. Therefore, participants as information producers took extra care to enable their new patches (commits) and cues (commit messages) to support the needs of future foragers (*consumers),* even if these efforts increased their own costs in their current task.

These attempts are consistent with the value/cost perspectives underlying IFT: participants aimed to increase the value, or decrease the costs of future foraging. For example, participants separated their changes into small, single-intent commits to reduce within-patch costs, they wrote detailed commit messages to help future foragers understand or estimate the value in commits, and, when they committed important changes, they notified their team about the availability of change awareness prey.

In spite of well-intentioned efforts like these, foraging difficulties due to nondescriptive commit messages and tangled commits persisted—they comprise the top two barriers in Table 3. Although it might be tempting to attribute these persisting barriers to "bad developer citizens" who did not adhere to good commit practices, our results reveal that there might be deeper underlying reasons.

7.2 Tensions: which future foraging need should a developer satisfy?

Different foraging situations might benefit from different characteristics in patches and cues. In our case, different future foraging situations placed conflicting demands of commits and commit messages, requiring participants to balance different kinds of tensions.

Tension #1: Short vs. Detailed Commit Message. Over 60% of our survey participants preferred detailed commit messages, to better estimate the value in a commit, or to understand the changes within it.

> S3: "High level statement of code changes with a detailed statement of the intent behind each change."

However, other participants found that detailed commit messages could take longer to read and process, especially when there were too many commits to read. To avert these costs, they preferred shorter and concise commit messages.

> P11: "Commit messages are often read in the command line application so they need to be very short."

Tension #2: Small vs. Large commits. Similar tensions surfaced in terms of commit sizes. Some participants preferred smaller commits with a single intent, to ease understanding of changes or to cherrypick specific changes.

> P6: "I try to keep all of my commits topical in nature. I try not to have different unrelated changes in the same commit… because that captures the history of development a little bit better. Not always, but it also makes it easier to prune out changes that were not necessarily beneficial.

However, other participants preferred larger commits (e.g., one commit per entire feature), to avoid fragmentation of related changes across commits, and to ease reviewing and merging code and for easier collaboration.

> P10: "I think that it helps reviewing because you can open the change set and you can see all the corresponding things that have changed as part of that change set. It is easier for the reviewers to coordinate that set of changes and pull them together ."

Tension #3: Current Task vs. Future Foraging. Sometimes a developer's current tasks needs were directly at odds with creating commits and commit messages for the future. For example, P2 attempted not to pollute his version control history with exploratory commits, but in doing so, he incurred additional costs from throwing away and reimplementing some changes:

> P2: "…I just threw the whole thing away… had I had finer grained commits where I could say this little part I would like and this little part… but I did not have the infrastructure to do that with just committing on the master branch."

In another instance, well-intentioned commit practices that aimed to ease future foraging actually hurt P11's ability to collaborate on a task:

> P11: "some people that feel that everything that is committed should compile and have running tests all the time… I had something that wasn't compiling that I needed to share with another developer."

Note that all these tensions are less about the preferences of individual developers, and more about costs and values in different foraging situations. Any attempt by the information producer to decrease the cost (or increase the value) for one foraging activity ended up disadvantaging another foraging activity, thereby making it hard for developers to balance these tensions and to meet the needs of *all* future foraging and current task activities.

> *Insight #5:* Participants' commit practices were largely sound: they aimed to increase the value or lower the costs for future foraging activities. But, they failed to balance the tensions between different foraging situations: 1) large vs. small commits, 2) detailed vs. concise commit messages and 3) current task completion vs. supporting future foraging.

Implication for tools: Unfortunately, IFT has little to offer by way of solutions for solving these tensions, but it offers some insights on how tools can improve costs and values for the future. For example, the design patterns catalog include patterns such as "rename methods" or "extract methods" that modify *existing* information in ways that could ease future foraging. The patterns could be lifted up to more general "extract patches" and "rename patches" to also include commits and commit messages. In fact, tools like HistoRef implement the extract patch idea for commits to refactor *existing commits*. But it is an open problem how tools should support creation of *new commit*s in VCS.

8 Discussion

In this section, we consider our results from a higher-level view.

8.1 IFT and the three-lens model

The three foraging activities we have considered in this paper add "hows" to the conceptual three-lens model presented in our earlier work [12] and summarized in Table 4. Specifically: 1) for the "awareness" lens, participants used lightweight foraging to learn about latest changes, 2) for the "archaeology" lens, participants looked for specific information using traditional foraging activities, and 3) for the "immediate" lens, participants looked into how they could create commits that will optimize costs and values for the future. These results not only ground the three-lens model in IFT's theoretical foundations, but also reveal new opportunities for tools in the awareness and immediate lens (e.g., bulk change awareness, balancing tensions), possible ways forward toward realizing those opportunities (namely, via IFT's design patterns) and some open research problems.

8.2 Open problem: IFT for change awareness

In change awareness foraging, participants were concerned with consuming as much high-value changes within limited time (or cost)—an optimization problem consistent with IFT. One mechanism in which they accomplished this optimization was by carefully choosing what prey to consume. Several tools and IFT patterns address this kind of problem—namely, helping developers choose which high value information to consume. However, limited support exists for another of their foraging mechanisms, namely lightweight foraging.

We advocate for research harvesting design patterns from existing tools in diverse change awareness domains to be added to the IFT design patterns catalog [34], as a way forward towards a more coherent body of practical knowledge about supporting people's information foraging. For example, our results revealed the reliance on commit messages or email subject lines (instead of entire commits) for change awareness foraging. This behavior is related to news foraging, where people heavily relied on headlines and summaries instead of reading entire news articles [49]. Such efforts provide benefits such as richer examples for the development of specific design patterns, and perhaps more importantly, help identify gaps in the design pattern catalog.

8.3 Open problem: the producer side

One of the new insights IFT brings to version control activities concerns information *producers*. Participants as information producers, recognized that the way information is created can impact how easy it will be to forage in that information in the future. Therefore, they attempted to create information in ways that will meet future foraging needs (even if that meant extra cost in creating that information).

Table 4. Relationship to the three-lens model.

Lens	Key activities	New insights from IFT
Awareness lens	Staying up to date with the recent changes	• Traditional dietary choices, suggestive of IFT's applicability to change awareness • Lightweight foraging, future vetting and future cost-value considerations different from traditional foraging. • Existing design patterns leverageable for selective, individual diet selection in tools. • **Open problem:** Extracting new design patterns from other change awareness domains to be added to Nabi et al.'s list [43]. • **Open problem:** Better understand cost/value aspects of lightweight and future-oriented foraging in change awareness.
Archaeology lens	Foraging for specific information, mostly in old commits	• Traditional foraging behaviors, suggesting IFT might apply to these activities. • Traditional IFT's improvements [40] (accurate cost/value estimation, improve actual cost/value estimates) to address participants' barriers to foraging. • Existing design patterns [43] leverageable for addressing barriers and improving tools.

Immediate lens	Creating commits and commit messages	• Creating patches and cues to increase value / decrease cost for future foragers.
		• Conflicting needs for different foraging situations, leading to tensions:
		  • Large vs. small commits
		  • Detailed vs. concise commit messages
		  • Current task completion vs. support future foraging activities
		• **Open problem:** Understand and balance tensions (e.g., via Social IFT) to build tools to support committing changes.

However, participants' attempts at producing forageable information were largely unsuccessful, because they failed to balance tensions arising from conflicting needs (e.g., between different foragers, between different future foraging situations or between current and future tasks). Balancing these tensions is important to meet diverse foraging needs of developers, yet IFT provides little intuition as to how developers (or tools) can achieve this balance.

Our results call for further enquiries into the producer side of information and how we can better support producers in creating forageable information. IFT's notions of costs and values could explain phenomena in these areas also, however, traditional IFT will not suffice.

Traditional foraging mostly treats predators to be solitary foragers: while this view sufficiently captures certain kinds of foraging in SE (e.g., individual developer foraging in code), it is not sufficient for collaborative situations (e.g., there is no construct for team). However, social IFT, a variant of IFT for cooperating groups, fills this gap.

Social IFT has allowed researchers to leverage the intelligence of the crowd in social tagging systems [44], to understand stakeholder interactions in requirements engineering [2] and to balance the tradeoffs between small vs. large open source projects to predict optimal team sizes [3]. Given that social IFT acknowledges, and has been applied to, tensions that arise in foraging situations, we believe that it might be a feasible framework for version control tensions also.

However, to make progress in this direction, research is needed to operationalize Social IFT to SE situations, given that Social IFT has not yet been widely operationalized. Also, Social IFT itself is nascent: therefore, research needs to start extending the theory's models for various collaboration situations as they arise in different domains. Addressing these issues could eventually inform tool builders in new ways about designing VCS tools for information producers.

8.4 Threats to validity

Every study has threats to validity.

One potential threat to external validity (the generality of our findings beyond this study [51]) is that our interviews included only 14 participants. We countered this threat in part by conducting a survey with 217 programmers from diverse backgrounds. Even so, the interview and survey responses could be biased due to limitations of human memory, the tools participants had used, and/or that their experiences probably did not cover all possible use cases. Therefore, our results might not generalize to all VCS tools, information-seeking strategies and situations.

Another potential threat to external validity is our study's focus on information foraging in *only one* environment, namely version control systems. Developers gather information from other environments also, and further studies would be needed to generalize our findings to other environments (e.g., bug repositories, emails) and to how developers synthesize information across them.

Another type of threat is reliability; a study is reliable if it yields the same results when conducted by other researchers [51]. One potential threat to reliability is the convenience and self selection (internet) sampling we used to recruit interview and survey participants respectively. Other researchers might not reach the same kinds of participants using these methods.

Another potential threat to reliability is that we re-analyzed data collected for a different purpose, not specific to IFT. As a result, some foraging-specific phenomena (e.g., cost-value considerations) might not have surfaced in our data, but might surface in a different study conducted with IFT in mind. Also, since we introduced IFT much later, we could not triangulate our IFT-based interpretations with the survey data. Only future studies leveraging IFT to VCS design can confirm the reliability of our findings.

Finally, this paper is a theoretical treatment of information seeking in version control systems; it does not implement or empirically evaluate the predictions and recommendations of the theory. However, this threat is partially addressed by other papers that have leveraged IFT's predictions for SE tool building (e.g., [38], [39], [53]).

9. Conclusion

In this paper, we have used an information foraging theory perspective to reveal new insights into developers' information seeking in VCS. Among the key results were:

- Participants engaged in traditional foraging behavior while foraging for specific commits. Here, IFT insights from earlier work can inform the design of VCS tools to support such foraging activities.
- To keep up with the latest changes, participants used lightweight foraging strategies to meet their highly selective and personalized information needs. This calls for change-awareness tools that provide low-cost affordances for less-detailed understanding of specific changes of interest.
- When committing their changes, participants attempted at the same time to ease future foraging activities. However, evidence suggests that they were not successful. One reason may be lack of awareness of the tensions involved in future foraging, such as individual vs. team, immediate vs. later, and different foraging strategies. This issue presents an open opportunity for VCS tools aiming to support developers' VCS foraging activities.
- Viewed from an IFT perspective, the participants' foraging adds "*hows*" to the "*whats*" in the conceptual three-lens model proposed in our earlier work [12], revealing implications for designing tools for each of these lenses.

Finally, our results provide some evidence that IFT's scope extends to the version control domain, but two open problems call for further research in this area. First, participants' lightweight change-awareness foraging was subtly different from traditional foraging; this calls for further inquiry into developers' lightweight foraging strategies for tasks like these. Second, our results reveal several "future-foraging" tensions developers were not successful in handling. These results open new avenues for research in the cost-value aspects of foraging in VCS environments, from both IFT and social IFT perspectives.

Acknowledgments

The authors wish to thank the reviewers; their feedback and suggestions greatly improved this paper. This work

was supported by DARPA #N66001-17-2-4030, NSF #1314384 and NSF CCF-1553741. Any opinions, findings, conclusions or recommendations expressed are the authors' and do not necessarily reflect the views of NSF, DARPA, the Army Research Office, or the US government.

REFERENCES

[1] A. Alali, H. Kagdi, and J.I. Maletic, "*What's a typical commit? a characterization of open source software repositories*," *Proc. Intl. Conf. Program Comprehension (ICPC '08)*, pp. 182-191, 2008.

[2] T. Bhowmik, N. Niu, P. Singhania, and W. Wang, "*On the role of structural holes in requirements identification: an exploratory study on open-source software development*," *ACM Trans. Management Information Systems (TMIS)* 6, no. 3 (2015): 10, doi:10.1145/2795235.

[3] T. Bhowmik, N. Niu, W. Wang, J.C. Cheng, L. Li, and X. Cao, "*Optimal group size for software change tasks: a social information foraging perspective*," *IEEE Trans. Cybernetics* 46, no. 8 (2016): 1784-1795, doi:10.1109/TCYB.2015.2420316.

[4] J.T. Biehl, M. Czerwinski, G. Smith, and G.G. Robertson, "*FASTDash: a visual dashboard for fostering awareness in software teams*," *Proc. SIGCHI Conf. Human factors in computing systems (CHI '07)*, pp. 1313-1322, 2007, doi:10.1145/1240624.1240823.

[5] C. Brindescu, M. Codoban, S. Shmarkatiuk, and D. Dig, "*How do centralized and distributed version control systems impact software changes?*" *Proc. 36th Intl. Conf. Software Engineering* (ICSE '14), pp. 322-333, 2014, doi:10.1145/2568225.2568322.

[6] Y. Brun, R. Holmes, M.D. Ernst, and D. Notkin, "*Proactive detection of collaboration conflicts*," *Proc. 19th ACM SIGSOFT Symp. and the 13th European Conf. Foundations of software engineering (FSE '11)*, pp. 168-178, 2011, doi:10.1145/2025113.2025139.

[7] J.L. Campbell, C. Quincy, J. Osserman, and O.K. Pedersen, "*Coding in-depth semistructured interviews: Problems of unitization and intercoder reliability and agreement*," *Sociological Methods & Research* 42, no. 3 (2013), pp. 294-320, doi:10.1177/0049124113500475.

[8] S.K. Card, J. Mackinlay, "*The structure of the information visualization design space*," Proc. IEEE Symp. *Information Visualization, 1997*, pp. 92-99, doi: 10.1109/INFVIS.1997.636792.

[9] E.H. Chi, J. Pitkow, J. Mackinlay, P. Pirolli, R. Gossweiler, and S.K. Card, "*Visualizing the evolution of web ecologies*," *Proc. SIGCHI Conf. on Human factors in computing systems (CHI '98)*, pp. 400-407, doi:10.1145/274644.274699.

[10] E.H. Chi, P. Pirolli, K. Chen, and J. Pitkow, "*Using information scent to model user information needs and actions and the Web*," *Proc. SIGCHI Conf. Human factors in computing systems (CHI '01)*, pp. 490-497, doi:10.1145/365024.365325.

[11] E.H. Chi, A. Rosien, G. Supattanasiri, A. Williams, C. Royer, C. Chow, E. Robles, B. Dalal, J. Chen, and S. Cousins, "*The bloodhound project: automating discovery of web usability issues using the InfoScentπ simulator*," *Proc. SIGCHI Conf. Human factors in computing systems (CHI '03)*, pp. 505-512.

[12] M. Codoban, S.S. Ragavan, D. Dig, and B. Bailey, "*Software history under the lens: a study on why and how developers examine it*," *IEEE Intl. Conf. Software Maintenance and Evolution (ICSME' 15), 2015*, pp. 1-10, doi:10.1109/ICSM.2015.7332446.

[13] L.F.C. Coy, M.L. Vásquez, J. Aponte, and D. Poshyvanyk, "*On automatically generating commit messages via summarization of source code changes*," *Intl. Working Conf. Source Code Analysis and Manipulation (SCAM), 2014*, pp. 275-284, doi:10.1109/SCAM.2014.14.

[14] C. de Souza, D.F. Redmiles, "*An empirical study of software developers' management of dependencies and changes*," *Proc. 30th Intl. Conf. Software engineering (ICSE '08)*, pp. 241-250.

[15] S.D. Fleming, C. Scaffidi, D. Piorkowski, M. Burnett, R. Bellamy, J. Lawrance, and I. Kwan, "*An information foraging theory perspective on tools for debugging, refactoring, and reuse tasks*," *ACM Trans. Software Engineering and Methodology (TOSEM)* 22, no. 2 (2013): 14.

[16] C. Gutwin, R. Penner, and K. Schneider, "*Group awareness in distributed software development*," *Proc. 2004 ACM conference on Computer supported cooperative work (CSCW '04)*, pp. 72-81, 2004.

[17] C. Gutwin, K. Schneider, D. Paquette, and R. Penner, "*Supporting group awareness in distributed software development*," *Intl. Workshop on Design, Specification, and Verification of Interactive Systems*, pp. 383-397, 2004, doi:10.1007/11431879_25.

[18] A. Guzzi, A. Bacchelli, Y. Riche, and A. van Deursen, "*Supporting developers' coordination in the IDE*," *Proc. 18th ACM Conf. on Computer Supported Cooperative Work & Social Computing*, pp. 518-532.

[19] S.Hayashi, D. Hoshino, J. Matsuda, M. Saeki, T. Omori and K. Maruyama, "*Historef: A tool for edit history refactoring*", *Proc. 22nd International Conference on Software Analysis, Evolution and Reengineering (SANER' 2015)*, pp. 469-473.

[20] A.Z. Henley, A. Singh, S.D. Fleming, and M.V. Luong, "*Helping programmers navigate code faster with Patchworks: A simulation study*," IEEE Symp. *Visual Languages and Human-Centric Computing (VL/HCC '14)*, pp. 77-80, 2014, doi: VLHCC.2014.6883026.

[21] A.Z. Henley, S.D. Fleming, and M.V. Luong, "*Toward Principles for the Design of Navigation Affordances in Code Editors: An Empirical Investigation*," *Proc. 2017 CHI Conf. Human Factors in Computing Systems* (CHI '17), pp. 5690-5702, doi:10.1145/3025453.3025645.

[22] P. Johnson, M. Ekstedt I. Jacobson, "*Where's the theory for software engineering?*", *IEEE software*, *29*(5), pp.96-96, 2012.

[23] B.K. Kasi, and A. Sarma, "*Cassandra: Proactive conflict minimization through optimized task scheduling*," *Proc. 2013 Intl. Conf. Software Engineering* (ICSE '13), pp. 732-741, 2013.

[24] D. Kawrykow, and M.P. Robillard, "*Non-essential changes in version histories*," *Proc. 33rd Intl. Conf. Software Engineering* (ICSE '11), pp. 351-360, 2011, doi:10.1145/1985793.1985842.

[25] S. Kim, E.J. Whitehead Jr, and Y. Zhang, "*Classifying software changes: Clean or buggy?*" *IEEE Trans. Software Engineering* 34, no. 2 (2008): 181-196, doi:10.1109/TSE.2007.70773.

[26] H. Kirinuki, Y. Higo, K. Hotta, and S. Kusumoto, "*Hey! are you committing tangled changes?*" *Intl. Conf. Program Comprehension* (ICPC 2014), pp. 262-265, doi: 10.1145/2597008.2597798.

[27] A.J. Ko, B.A. Myers, M.J. Coblenz, and H.H. Aung, "*An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks*," *IEEE Trans. software engineering*, 32, no. 12 (2006), doi: 10.1109/TSE.2006.116.

[28] A.J. Ko, R. DeLine, and G. Venolia, "*Information needs in collocated software development teams*," *29th Intl. Conf. Software Engineering ICSE '07)*, pp. 344-353, 2007, doi:10.1109/ICSE.2007.45.

[29] S.K. Kuttal, A. Sarma, and G. Rothermel, "*Predator behavior in the wild web world of bugs: An information foraging theory perspective*," *IEEE Symp. Visual Languages and Human-Centric Computing (VL/HCC '13)*, pp. 59-66, doi: 10.1109/VLHCC.2013.6645244.

[30] J. Lawrance, R. Bellamy, and M. Burnett, "*Scents in programs: Does information foraging theory apply to program maintenance?*" *IEEE Symp. Visual Languages and Human-Centric Computing (VL/HCC '07)*, pp. 15-22.

[31] J. Lawrance, R. Bellamy, M. Bumett, and K. Rector, "*Can information foraging pick the fix? A field study*," *IEEE Symp. Visual Languages and Human-Centric Computing (VL/HCC '08)*, pp. 57-64.

[32] J. Lawrance, M. Burnett, R. Bellamy, C. Bogart, and C. Swart, *"Reactive information foraging for evolving goals," Proc. SIGCHI Conf. Human Factors in Computing Systems (CHI '10)*, pp. 25-34.

[33] M. Marzban, Z. Khoshmanesh, and A. Sami, *"Cohesion between size of commit and type of commit,"* In *Computer Science and Convergence*, pp. 231-239, 2012, doi:10.1007/978-94-007-2792-2_22.

[34] T. Nabi, K. Sweeney, S. Lichlyter, D. Piorkowski, C. Scaffidi, M. Burnett, and S. D. Fleming, *"Putting information foraging theory to work: Community-based design patterns for programming tools," IEEE Symp. Visual Languages and Human-Centric Computing* (VL/HCC'16), pp. 129-133.

[35] S. Negara, M. Vakilian, N. Chen, R.E., Johnson, R. E. and D. Dig, "Is it dangerous to use version control histories to study source code evolution?" Proc. *European Conference on Object-Oriented Programming* (ECOOP'12), pp. 79-103.

[36] A.T. Nguyen, M. Hilton, M. Codoban, H.A. Nguyen, L. Mast, E. Rademacher, T. N. Nguyen, and D. Dig, *"API code recommendation using statistical learning from fine-grained changes," Proc. 24th ACM SIGSOFT Intl. Symp. Foundations of Software Engineering (FSE '16)*, pp. 511-522, doi: 10.1145/2950290.2950333.

[37] N. Niu, A. Mahmoud, Z. Chen, and G. Bradshaw, *"Departures from optimality: understanding human analyst's information foraging in assisted requirements tracing," Proc. of the 2013 Intl. Conf. Software Engineering (ICSE '13)*, pp. 572-581.

[38] A. Perez and R. Abreu, "A diagnosis-based approach to software comprehension," *Proc. 22nd Intl. Conf. Program Comprehension (ICPC '14)*, pp. 37-47, doi:10.1145/2597008.2597151.

[39] D. Piorkowski, S. Fleming, C. Scaffidi, C. Bogart, M. Burnett, B. John, R. Bellamy, and C. Swart, *"Reactive information foraging: An empirical investigation of theory-based recommender systems for programmers," Proc. SIGCHI Conf. Human Factors in Computing Systems (CHI '12)*, pp. 1471-1480, doi: 10.1145/2207676.2208608.

[40] D. Piorkowski, S. D. Fleming, I. Kwan, M. M. Burnett, C. Scaffidi, R. Bellamy, and J. Jordahl, *"The whats and hows of programmers' foraging diets," Proc. SIGCHI Conf. Human Factors in Computing Systems (CHI '13)*, pp. 3063-3072, doi:10.1145/2470654.2466418.

[41] D. Piorkowski, A.Z. Henley, T. Nabi, S.D. Fleming, C. Scaffidi, M. Burnett, *"Foraging and navigations, fundamentally: developers' predictions of value and cost," Proc. 2016 24th ACM SIGSOFT Intl. Symp. Foundations of Software Engineering (FSE '16)*, pp. 97-108.

[42] P. Pirolli, and S. Card, *"Information foraging in information access environments," Proc. SIGCHI Conf. Human factors in computing systems (CHI '95)*, pp. 51-58, doi: 10.1145/223904.223911.

[43] P. Pirolli, and S. Card, *"Information foraging," Psychological review* 106, no. 4 (1999): 643.

[44] P. Pirolli, *Information foraging theory: Adaptive interaction with information*, Oxford University Press, 2007.

[45] S.S. Ragavan, S.K. Kuttal, C. Hill, A. Sarma, D. Piorkowski, and M. Burnett, *"Foraging among an overabundance of similar variants," Proc. 2016 Conf. Human Factors in Computing Systems (CHI '16)*, pp. 3509-3521, doi: 10.1145/2858036.2858469.

[46] S.S. Ragavan, B. Pandya, D. Piorkowski, C. Hill, S.K. Kuttal, A. Sarma, and M. Burnett, *"PFIS-V: Modeling Foraging Behavior in the Presence of Variants," Proc. of the 2017 Conf. on Human Factors in Computing Systems*, pp. 6232-6244, doi:3025453.3025818.

[47] R. Robbes, and M. Lanza, "A change-based approach to software evolution" *Electronic Notes in Theoretical Computer Science*, *166*, 93-109. (2007).

[48] A. Sarma, D.F. Redmiles, and A.V.D. Hoek, *"Palantir: Early detection of development conflicts arising from parallel code changes," IEEE Trans. on Software Engineering* 38, no. 4 (2012): 889-908.

[49] A.J. Sellen, R. Murphy, and K.L. Shaw, *"How knowledge workers use the web," Proc. SIGCHI conference on Human factors in computing systems (CHI '02)*, pp. 230-234, doi:10.1145/503376.503418.

[50] S. Shivaji, E.J. Whitehead, R. Akella, and S. Kim, *"Reducing features to improve code change-based bug prediction," IEEE Trans. Software Engineering* 39, no. 4 (2013): 552-569.

[51] F. Shull, J. Singer, and D. I. Sjøberg, *"Guide to advanced empirical software engineering"*, 2008.

[52] J. Sillito, G.C. Murphy, and K.D. Volder, *"Questions programmers ask during software evolution tasks," Proc. 14th ACM SIGSOFT Intl. Symp. Foundations of software engineering (FSE '06)*, pp. 23-34

[53] J.M. Spool, C. Perfetti, and D. Brittan, *"Designing for the Scent of Information: The Essentials Every Designer Needs to Know About How Users Navigate Through Large Web Sites," User Interface Engineering* (2004).

[54] D.W. Stephens, and J.R. Krebs, *Foraging theory*, Princeton University Press, 1986.

[55] Y. Tao, Y. Dang, T. Xie, D. Zhang, and S. Kim, *"How do software engineers understand code changes? : an exploratory study in industry," Proc. Intl. Symp. on the Foundations of Software Engineering (FSE '12)*, p. 51.

[56] M.B. Zanjani, G. Swartzendruber, and H. Kagdi, *"Impact analysis of change requests on source code based on interaction and commit histories," Proc. 11th Working Conf. Mining Software Repositories*, pp. 162-171, 2014, doi:10.1145/2597073.2597096.

[57] S.S. Ragavan, *"Software History"* at https://web.engr.oregonstate.edu/~srinivas/software-history.html, 2018.

Sruti Srinivasa Ragavan is a Ph.D candidate at Oregon State University. Her research interests lie at the intersection of Human Computer Interaction and Software Engineering, in leveraging human behavioral theories to understand and support programmers. Prior to graduate school, she worked as a senior software developer at ThoughtWorks, building bespoke solutions for various businesses.

Mihai Codoban received the B.S. in Computer and Information Technology from Politehnica University of Timisoara in 2011, and the M.S. in Computer Science from Oregon State University in 2015. He joined Microsoft in 2015 where he is working on build engines as part of the Tools for Software Engineers group. His interests are in understanding programmers' needs and in building tools to fulfill those needs.

David Piorkowski received his PhD in Computer Science from Oregon State University in 2016. He is a Research Staff Member at IBM Research. His research interests focus on understanding and supporting developers' debugging tasks. Recently he's been working on understanding how software developers cope with AI models. He is a member of the IEEE.

Danny Dig is an associate professor of computer science at Oregon State University, and an adjunct professor at University of Illinois. He pioneered interactive program transformations and opened the field of refactoring to new domains such as mobile, component-based, and end-user programming. He earned his Ph.D. from the University of Illinois at Urbana-Champaign where his research won the best Ph.D. dissertation award, and the First Prize at the ACM Student Research Competition Grand Finals. He did a postdoc at MIT.

Margaret Burnett is an OSU Distinguished Professor at Oregon State University. She co-founded the area of end-user software engineering, pioneered the use of information foraging theory in software debugging, and leads the team that created GenderMag. Burnett is an ACM Fellow, a member of the ACM CHI Academy, and an award-winning mentor. She also serves on the Academic Alliance Advisory Board of the National Center for Women In Technology (NCWIT).