AN ABSTRACT OF THE THESIS OF

Sruti Srinivasa Ragavan for the degree of Master of Science in Computer Science presented on January 10, 2018.

Title: Version Control Systems: An Information Foraging Perspective.

Abstract approved:

_____

Margaret M. Burnett

Software history and version control systems (VCS) are an important source of information for developers. This importance entails the need for a principled understanding of developers' information seeking in VCS, both for improving existing tools as well as understanding requirements for new tools. However, it is only recently that researchers have started investigating *how* developers use VCS.

In this thesis, we take a theory-based approach to understanding developers' information seeking in VCS. Using the foundations of Information Foraging Theory (IFT), we analyze the data from a prior empirical study, to gain new perspectives into developers' information seeking in VCS. Our results indicate that participants engaged in foraging behavior; therefore, tool builders can leverage IFT's design insights and patterns to VCS design.

Further, our results reveal that participants' change-awareness foraging differed subtly from traditional foraging and calls for further investigation. Similarly, participants attempted to create commits that suited the needs of future foragers. However, balancing the tensions between different foraging activities or between different people (e.g., different commit size preferences) is a hard problem and presents an opportunity for further research at the intersection of IFT and software engineering.

©Copyright by Sruti Srinivasa Ragavan
January 10, 2018
All Rights Reserved

Version Control Systems: An Information Foraging Perspective

by
Sruti Srinivasa Ragavan

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented January 10, 2018
Commencement June 2018

Master of Science thesis of Sruti Srinivasa Ragavan presented on January 10, 2018

APPROVED:

Major Professor, representing Computer Science

Director of the School of Electrical Engineering and Computer Science

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Sruti Srinivasa Ragavan, Author

ACKNOWLEDGEMENTS

First of all, I'd like to thank my major advisor, Dr. Margaret Burnett, who not only taught me to ask questions and provided great research and writing feedback, but also went out of her way to support and encourage during some low phases. But for her perceptive eyes, patient ears, kind words and immense understanding, this thesis wouldn't have seen the light of the day. I owe it all to you, Dr. B; someday, I'll grow up to be like you.

I'd also like to thank my committee members, with whom in-class as well as out-of-class interactions have been fun and insightful. Thank you Dr. Anita Sarma, Dr. Eric Walkingshaw, Dr. Cindy Grimm and Dr. David Kling.

Before joining Margaret's lab, I worked in Dr. Danny Dig's lab, where I did some of the initial work building up to this thesis. Thank you for your continued encouragement and support, Danny.

I'd also like to thank my grad school buddies for challenging me at work, giving me the good hard times, putting up with my quirks, providing advice and for making life fun in general. Bhargav Pandya, Amin Alipour, David Piorkowski, Mihai Codoban and Michael Hilton, I miss you all! Sharmin Sultana and Kamala Sadagopan Raghavan, thank you both for being awesome roommates.

Finally, I'd like to thank my family and friends for encouraging me to go to grad school, for staying up and skyping with me at wee hours and for just being there. I don't know what I'd do without you all.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

LIST OF TABLES

Table Page

1 INTRODUCTION

Software engineering (SE) is an information-intensive activity. Empirical studies have revealed that developers ask and seek answers to several questions as part of their day-to-day development activities [44, 24, 25]. In order to satisfy their information needs, that range from questions about existing code to collaboration needs, developers turn to various information sources such as bug repositories, documentation, web or even other team members. One such source of information for developers is the software history that resides in the project's version control system (e.g., Git, SVN, Hg) [25].

Version control systems (VCS) are a rich source of information and SE researchers have leveraged this source to gain insights into various aspects of software engineering, such as predicting bugs [44], predicting merge conflicts [6] and recommending APIs [32], to name a few. However, VCS are also an important source of information for developers [25]; yet, surprisingly little research has focused on how developers use the information in VCS. Our recent work was the first to characterize the *whats*, *whys*, *hows* and barriers to developers' information seeking in VCS [11].

In this thesis, we bring a new perspective—that of the Information Foraging Theory (IFT)—to developers' information seeking in VCS. Based on the idea that the information-seeking behavior is similar to the food-foraging behavior of animals, IFT has successfully explained the *hows* and *whys* of people's information seeking in various domains, including document collections, web and software engineering [39, 38, 27, 33, 40, 26]. Encouraged by these prior successes, we chose IFT as the theoretical framework to systematically understand developers' information seeking in VCS. Towards this end, we present an IFT-based analysis of the data from our prior study [11].

We also chose IFT because of its practical applicability. IFT has informed the design and evaluation of various information environments such as websites and web search engines [9, 10], complex visualizations [8] and SE tools [34, 19]. From these design insights, researchers have further distilled –via the underlying IFT's constructs and propositions– design principles and patterns, such as web-design guidelines [47], principles for IDE navigations [20, 36] and design patterns for SE tools [15, 31]. By framing

developers' foraging in VCS using the IFT vocabulary, tool builders can leverage the commonalities between the VCS domain and the existing IFT design principles and patterns. For example, tool builders can systematically reuse existing IFT-based design solutions in VCS environments, rather than reinventing the wheel.

The rest of this thesis is organized as follows. Chapter 2 presents a review of the related work on IFT and information seeking in VCS. Chapter 3 provides a brief IFT primer and maps the VCS domain to IFT's constructs. Chapter 4 describes the methodology of the prior study whose data we re-analyze. Chapter 5 discusses the key results: (i) how developers forage for specific commits, (ii) how they keep up with the latest changes happening on the project and (iii) how they create commits to ease future foraging activities. This chapter also discusses how tool builders can draw upon existing IFT-based design patterns to inform the design of VCS environments. Chapter 6 builds upon the results in Chapter 5 and relates them to the three-lens model of history we proposed in our prior work [11]. The chapter also highlights several gaps in existing research and discusses future research opportunities at the intersection of SE and IFT. Finally, Chapter 7 concludes the thesis with a brief summary of the key takeaways.

2 RELATED WORK

For this thesis, we reviewed the literature in two areas: information foraging theory and version control systems.

2.1 Information Foraging Theory

Information Foraging Theory (IFT) was first defined by Pirolli and colleagues to explain people's information seeking behavior in large document collections [37], who then used the theory to explain experts' information seeking [38] and information visualizations [8]. They then applied IFT extensively to web foraging, such as to explain people's web browsing behavior [9] and to inform and evaluate the design of websites [10], subsequently laying the foundations for web design [47].

In software engineering, Ko et al. first suggested IFT as an underlying theory for information-seeking in SE [24]. Ever since, researchers have applied IFT to explain information-seeking in various SE tasks, including requirements engineering [33] and debugging [28, 29]. Recent work by our group has also looked at programmers' foraging in the presence of program variants in an exploratory programming context [40, 41]. Most of this work falls into two categories—namely, empirical studies of programmer's behavior or using IFT-based computational models to predict programmers' navigations.

Based on these studies and models, researchers have also built several practical tools to aid programmers' foraging. Piorkowski et al. built a recommendation system that recommends the next location a programmer should navigate to [34] while Henley et al. built the Patchworks code editor to ease programmers' foraging [19].

In addition to specific tools, researchers have also attempted to crystallize the theory's propositions into generic IFT-based design patterns for the design of SE tools. Fleming et al. examined several research and commercially-available tools and proposed a set of initial design patterns [15]. Nabi et al. then extended this initial set into a crowd-based actively-growing catalog [31]. However, none of these studies consider developers' foraging in VCS environments.

2.2 Version Control Systems

Version control systems (VCS) have been in use in the software industry over the last three decades; yet, not much related work exists on developers' information-seeking in version control systems. Our prior work [11], on which this thesis is built, was one of the first studies that characterized developers' usage of VCS and software history—the why, the how, and the problems they encountered.

However, as the rest of this section outlines, researchers have studied specific VCS-related activities and proposed solutions to specific problems that developers encountered. We frame this body of related work based on the three foraging activities, namely, change awareness, locating specific information and creating commits, described later in Chapter 5.

2.2.1 Change awareness

A large body of literature exists on collaboration in software development, which also includes change awareness. Several empirical studies have characterized various awareness needs of developers, including impact management, conflict management and awareness needs. For example, Gutwin et al. studied developers' change awareness on open-source projects and classified them into general awareness and specialized awareness [16], while Guzzi et al. investigated the collaboration practices of developers in large teams [18]. Similarly, DeSouza et al. empirically characterized developers' impact management as forward and backward impact management [10].

Based on these empirical findings, researchers have built several tools to meet developers' collaboration needs. For example, Gutwin et al. built ProjectWatcher to aid developers' general awareness [17], while Guzzi et al. recently built Bellevue to facilitate change awareness within the IDE [18]. Cassandra, Palantir and FastDash are some examples of tools that help developers minimize conflicts [42, 21, 4].

This thesis complements the existing body of work on change awareness and takes a theoretical approach to understanding how developers forage for change awareness in VCS. By using IFT's cost-value proposition (discussed in detail in Chapter 3), we provide

deeper insights into developers' change-awareness foraging behavior, which in turn can inform the design of VCS to ease awareness as well as awareness tools.

2.2.2 Locating specific commits

Contrary to change awareness, not much related work exists about *how* developers seek answers to specific task-related questions in VCS. However, empirical studies of developers (e.g., [46] [25]) have suggested generic ways to improve information-seeking for developers (e.g., aggregate all information related to a bug), including gathering information from VCS. Similarly, Tao et al. studied how developers understood changes in commits [48] and revealed the limitations of existing tools and practices. This thesis goes beyond just understanding commits and focuses on information-seeking in VCS in depth, taking a theoretical approach.

2.2.3 Committing changes

In the realm of committing changes, a lot of related work has focused on studying specific characteristics of commits. Alali et al. investigated the characteristics of a typical commit in open-source projects [1], while Kawrykow and Robillard found that up to 26% of all source-code modifications were minor (e.g., renaming local variable) [22]. Other studies, such as the one by Marzaban et al. [30], have attempted to predict the nature of changes in a commit using commit sizes. Brindescu et al. compared the size of commits across Git and SVN users and also observed the phenomenon that developers had different commit size preferences [5].

Similarly, in the case of commit messages, researchers have focused on automatically generating commit messages [12], while others have analyzed commit messages to predict the nature of the changes [23] or the impact of incoming change requests [49].

This thesis differs from prior work and takes a theoretical approach to creating commits and commit messages. We explore developers' considerations about future foraging activities and the tradeoffs involved in satisfying different needs for different foraging activities or different foragers.

3 BACKGROUND: INFORMATION FORAGING THEORY

Information Foraging Theory (IFT) is a theory of how people seek information. Derived from the optimal foraging theory rooted in biological sciences, IFT posits that people seek information in ways similar to how predators forage for their prey in the wild. Pirolli and Card first used IFT to explain how experts foraged for information in document collections [37]. Since then, IFT theory has been applied to several domains—to explain people's information-seeking behaviors or to inform the design of environments.

Construct	**Definition**	**Example (in VCS)**
Predator	Person foraging for information	Developer
Prey	Information that the predator is seeking	Location of a bug
Information environment	Environment where the foraging happens	Version control system
Patches	Locations in information environment	Commit, list of commits
Links	Connection between patches	Hyperlinks from one commit to previous and next commits.
Cues	Hints about information at the other end of a link	Words in commit messages, file names.
Scent	Predator's estimate of the information value at the other side of a link	Similar words = higher scent (Note: scent is in the predator's head)

Table 1 Constructs of the Information Foraging Theory.

IFT uses a small set of constructs derived from the optimal foraging theory to explain people's information seeking. Table 1 summarizes these constructs, along with an example operationalization in the VCS domain. The *predator* refers to the person seeking information (e.g., software developer), while the *prey* refers to the information the predator is seeking (e.g., location of a bug). The predator forages for its prey within an *information environment* (here, version control systems), which is analogous to the foraging grounds in the wild. Figure 1 shows the information environment of gitk, the built-in GUI repository browser for Git. Other popular version control systems (e.g., SVN, Hg), as well as third-party VCS tools (e.g., TowerGit, GitHub) also provide similar information environments.

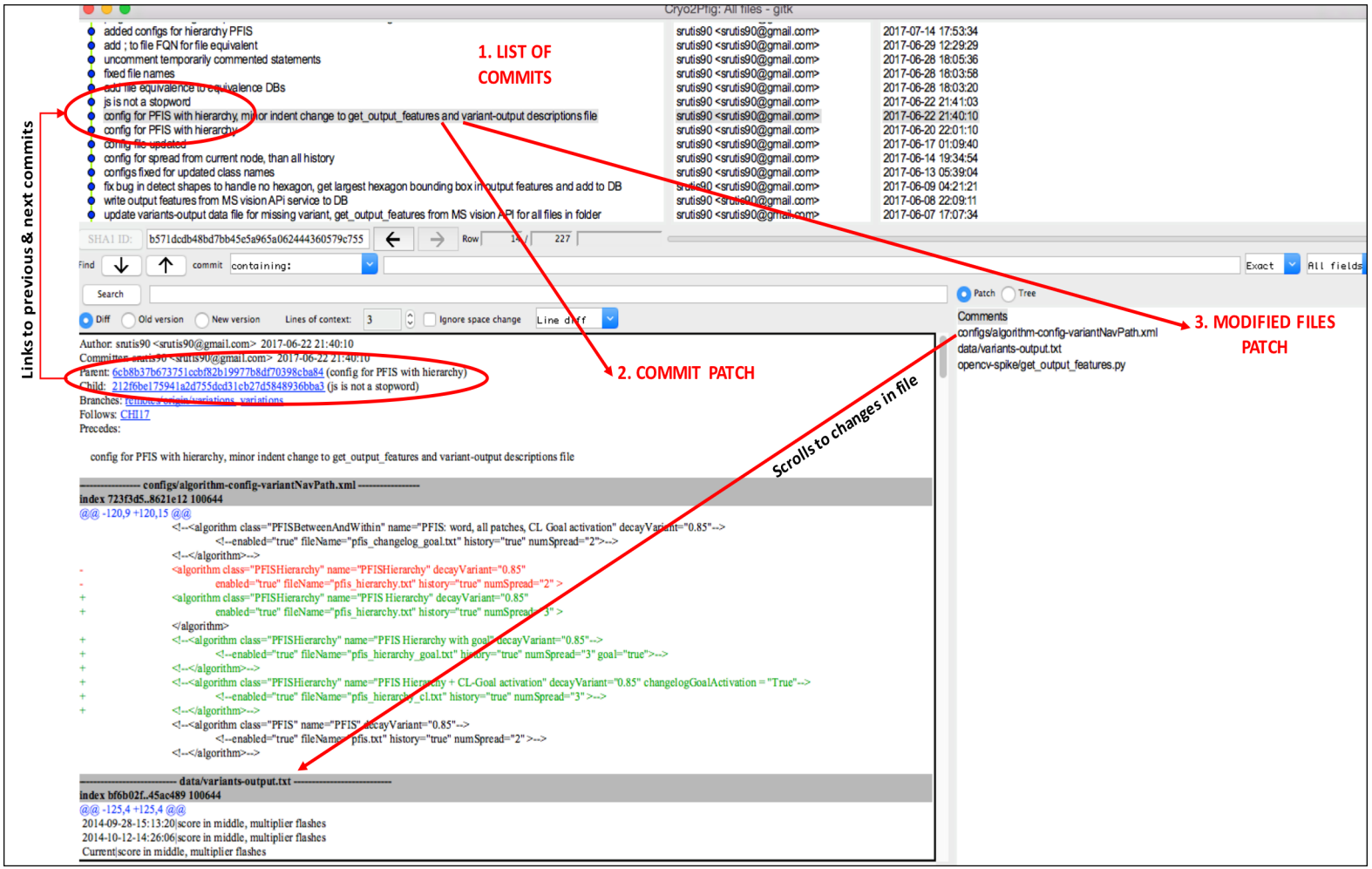

Figure 1 VCS Information Environment. The environment is organized as patches (numbered labels), , connected by links (red arrows) .

The information environment consists of information *patches*, connected via *links*. For example, in Figure 1, each commit is a patch: label 2 shows a single commit patch, which is linked to its previous and next commits via the "Parent" and "Child" links (the red arrows in the figure represent links). The list of all commits (labeled 1) is also a patch, where each row is a link: clicking on the link opens the corresponding commit and the modified files in Patches 2 and 3 below, as the red arrows from Patch 1 to Patches 2 and 3 show. Similarly, the list of modified files (labeled 3) also contains links, where clicking on a file name in Patch 3 scrolls to the changes for that file in Patch 2.

Notice that the links above contained labels such as commit messages in Patch 1 or file names in Patch 3: these are called *cues* and provide hints about where a link might lead to and what information might be found at the other end of the link. For example, the word "configure" in the commit message indicates that there might be some changes made to the configurations file. A predator interprets these cues in the environment and uses the *scent* to decide which links to take (or not), in order to hunt down its prey: this is similar to how animals sniff at cues (e.g., hoof prints) and use the *scent* to guide them to their prey. Note that, unlike cues, scent is not in the information environment: instead it is in the predator's head.

More formally, IFT's fundamental assumption is that the predator will try to forage for the prey in an optimal manner. Therefore, in an environment where a predator has several choices, with different costs and information gains associated with each action, IFT posits that the predator makes foraging choices in such a way that maximizes the *rate of information gain.* In other words, the predator's foraging choices attempt to maximize the information value, V, gained per unit expended cost, C, as characterized by the equation:

$$Predator's\ choice = Maximum\left(\frac{V}{C}\right) \quad (1)$$

However, most of the time, the predator cannot predict beforehand the actual value or cost associated with an action. Instead, the predator's choices based on their expectations of the values and costs: this value-to-cost estimate is called the *information scent*. Therefore, the equation (1) can be revised as

$$Predator's\ choice = Maximum\left(\frac{E(V)}{E(C)}\right) = Maximum\ scent \quad (2)$$

E(V) and E(C) denote the expected value and expected cost respectively. In the above equations, the cost refers to the cost of performing an action, which includes cost of processing cues, the cost of following links or costs associated with processing information. The cost might be measured in terms of number of mouse clicks, time taken or the cognitive effort involved.

This cost-value proposition of IFT, characterized by equations (1) and (2), are central to leveraging IFT for designing information environments. On one hand, these propositions allow researchers to predict and explain what a user will do in an information environment, thereby uncovering any existing design flaws. On the other hand, these constructs and propositions are generic; therefore, researchers have extracted generic insights for designing optimal environments that can be applied to several domains. For example, Piorkowski et al. recently highlighted that environments can ease predators' foraging by helping them align their cost and value estimates with the environment's actual costs and values. Similarly, Fleming et al. [15] and Nabi et al. [31] leveraged the commonalities in SE tools and cataloged them into IFT-design patterns. Since the problems and the solutions are both phrased in generic IFT's terms, instead of specific environments, such design patterns and solutions are easily transferable to other SE environments such as version control systems (as we shall see later).

4 METHODOLOGY

In this thesis, we re-analyze the data –using a new perspective– from our prior study on developers' usage of software history [11]; therefore, in this chapter, we discuss the methodological details of that original study (also described in [11]).

The study involved two stages: first, we conducted interviews with professional developers to elicit why and how developers used software history and any barriers they faced while doing so. We then conducted a survey to quantify the findings from the interviews. The survey also provided a way of triangulating the findings from the interviews.

4.1 Interviews

We interviewed 14 software developers from 11 companies, with an average experience of 13 years. They used diverse VCS (Git, SVN, TFS, Bazaar) and clients (command line, Github, Stash, etc.). Each interview lasted between 40 and 90 minutes and we paid the participant $50 at the end of the interview. We used semi-structured interviews: we started with a fixed set of questions to structure the interview; based on participants' responses, we asked follow-up questions to gather additional details. Such a technique provided a framework to keep all the interviews focused on the same topics while also allowing interesting tangents to be followed [45]. Figure 2 shows the initial fixed set of questions that we used as the framework for our interviews.

First, we transcribed the interviews. Then, we split each interview up into small segments to be individually coded: this is called segmentation. As Campbell et al. note, the segmentation required "subjective interpretation, contextualization, and especially a thorough understanding of the theoretically motivated questions guiding the study" [7]. In our case, the interviews were conducted by a second researcher (different from the author): therefore, the second researcher performed the segmentation. Specifically, we segmented the transcript every time a participant changed the topic of the subject or a new question was asked.

1. Can you describe the current software project that you are working on?
2. What is your role in the company / project?
3. What VCS tools are you using? What GUI clients do you use for the VCS tool?
4. Can you describe a recent, but completed software change that you made?
    a. When was the change made?
    b. How was the change recorded or archived?
    c. How many other developers were involved in carrying out this change?
    d. How did you coordinate with them in making this change?
    e. How did you communicate to others that the change was made?
    f. What was the most difficult aspect of making the change?
    g. What sort of tools or processes would have made it easier to help with this task?
5. Can you describe an old software change that you made? (details similar to question #4)
6. Can you describe an instance when you had to access a recent change to the software?
    a. What was the purpose of accessing the change?
    b. How did you do it? (strategies, tools, people, extra info)
    c. What was the most difficult aspect of accessing the change?
    d. What sort of tools or processes would have made it easier to help you with this task? (If you had a magic wand to make the code repositories more effective for you, how would you make them?)
7. Can you describe an instance when you had to access an older (not what you would consider recent) change to the software? (details similar to question #6)
8. What is the most important reason for which you access prior changes to the software?
9. What are the biggest challenges you face when you when accessing history?
10. How do you gain awareness of the latest or important changes made to the software? For example, do you ask others or simply browse the repository?

Figure 2 Initial set of questions for the semi-structured interviews. We followed interesting tangents based on participants' responses.

The author and the second researcher then coded the transcripts, proceeding one interview at a time. The coding session for each transcript proceeded as follows. First, the two coders independently coded each segment, allowing multiple codes per segment. Since no prior code set on VCS usage was available, the coders adopted open coding and coded participants' motivations, strategies and barriers in using VCS.

After the independent coding, the coders compared their independent code sets. Often, the two coders gave different names for the same code; therefore, they renamed the codes

to be consistent with each other. For example, one coder had a code named "*why is this this way*", while the other coder called the code "change rationale"; when they compared the code sets, the coders renamed the code to "why is this *this* way" in the code set as well as in coded transcripts. Such renaming resulted in inter-rater agreement (IRR) of 65% (Jaccard index) averaged across all sessions. These IRR levels are consistent with the measures that Campbell et al. report [7].

Then, the coders resolved their coding disagreements using the negotiated agreement technique [7]: they resolved their disagreements by mutually agreeing upon and adding new codes, deleting and merging existing codes or disambiguating the description of codes. Whenever a code was thus modified (including codes combined, merged or deleted), the coders also revisited previously coded interviews and re-coded the segments containing instances of the modified codes. Following this method, the coders got a final IRR of 97.4%, averaged across all participants.

By following such a process, at the end of each session the coders had a common agreed-upon code set based on the interviews coded so far. They used this agreed-upon code set from the previously-coded interviews as the starting code set for coding the next participant's transcript. It took 10 interviews to stabilize the code set using this process, consistent with prior studies on semi-structured interviews [7]. However, the coders continued to code the remaining interviews in a similar manner. Once the coding was complete for all interviews, we grouped the codes into larger emerging themes.

4.2 Surveys

Interviews provided rich qualitative data about VCS usage, but only from a small sample size. Therefore, to validate our data with a broader demographic as well as to quantify our interview findings, we designed a survey. Figure 3 shows the survey questions, which we derived from the interview findings. Since the sample size of the interview participants was limited, we included "*other*" fields for all multiple-choice questions, in case the survey respondents had additional insights beyond the ones we derived from the interviews. The complete survey can be found on this study's companion website [50].

We recruited 217 survey respondents by advertising our survey on social media channels (Reddit and Twitter) frequently accessed by software developers. 80% of the respondents had more than 5 years of professional experience, and 84% were currently practitioners from the industry.

1. How important is software history for your development activities?
2. How frequently do you access software history during your development activities?
3. When accessing software history, how frequently do you access the following types of code changes?
4. What are the most important reasons you access recent history for? Choose the top reasons (up to 3).
5. What are the most important reasons you access old history for? Choose the top reasons (up to 3).
6. Other than the reasons mentioned above, do you have additional reasons you access software history for?
7. How do you gain awareness about what recent code changes were made on your project? Choose all that apply.
8. For each of the following recent code changes, how important is it for you to become aware of them as soon as possible?
9. How frequently do you use the following (strategies) to understand the intent of a commit?
10. Which of the following (barriers) do you have difficulties with when accessing history? Choose the most significant ones.
11. Which of the following (features) would be most helpful for you when working with software history? Choose the most significant ones.
12. What information would you like an ideal commit message to contain?
13. Overall, how satisfied are you with the current tool support for working with software history?
14. What is the extent of your programming experience?
15. What type of project do you spend the majority of your development time on?
16. Which of the following best defines your role for the past year?

Figure 3 Survey questions. The survey contained multiple choice questions that were directly derived from the interviews.

In this thesis, we present a fresh interpretation of the data from the prior study using an IFT-based perspective. To do so, we mapped the code-set from the prior study to IFT's constructs and propositions. Specifically, we mapped developers' motivations in VCS to

foraging goals and framed their strategies and barriers based on IFT's constructs and propositions.

5 RESULTS: AN IFT PERSPECTIVE

Participants engaged in three major foraging activities in VCS, namely foraging for change awareness, foraging for specific commits and creating commits to ease future foraging. We frame our results on developers' information foraging for each of these activities.

5.1 Change awareness: lightweight foraging

One of developers' information needs on a project is to keep up with the latest changes happening on the project [25]. Over 70% of our survey participants foraged in version control systems to gain such change awareness.

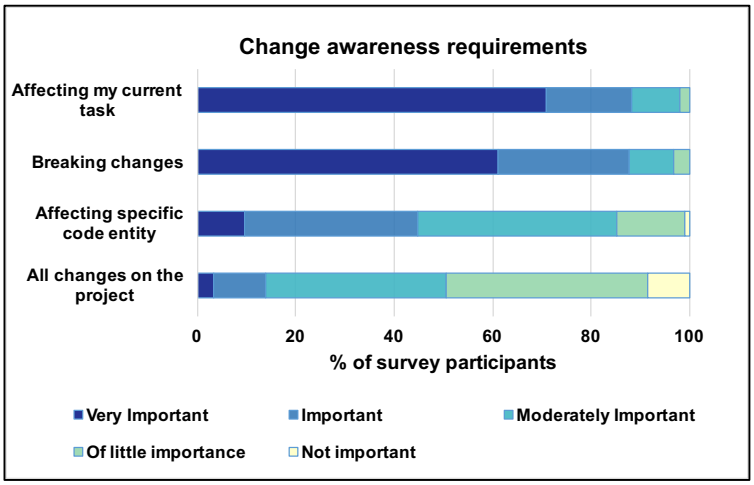

Figure 4 Change awareness requirements. Participants did not want to learn about all changes happening on the project; instead, their prey was selective and personalized.

However, as Figure 4 shows, participants did not want to keep up with all the changes happening on their project

> P12: *"I do not read every single commit that goes through my codebase… If something does not look as [if] it is needed I shall ignore it".*

In other words, participants did not consume all the prey that was available. Instead, participants' prey was highly selective (e.g., breaking changes) and personalized (e.g., changes affecting *my* current task, or changes to code that is of interest to *me*).

Such change-awareness foraging was subtly different from the traditional notions of foraging. Traditional IFT has primarily focused on foraging in environments where the information mostly remains unchanged or is changed only by the predator (e.g., programming in changes while fixing a bug) [29]. However, change-awareness foraging presents a situation where changes to the information environment are constantly introduced by several people (including the predator). This key difference manifested as differences in foraging behaviors between traditional and change-awareness foraging situations.

The first difference was in the motivation for the foraging. Traditionally, IFT assumed that "*information foraging is usually a task that is embedded in the context of some other task*", i.e., the motivation as task completion. However, change-awareness foraging was not directly motivated by task completion but was a response to the changing nature of the information environment. By keeping up with the latest changes, participants attempted to minimize future costs, such as preventing bugs or easing future foraging by maintaining an up-to-date mental model of the project. For example, P4 wanted to keep up with the latest changes to avoid potential merge conflicts:

> P4: *"If I think that there is nothing to fear, then I just do a CVS update and merge the previous checks to bring other people's changes into my view of the world. If I think that I need to be more cautious then I'll do a TkDiff, looking at the files that are most crucial, see what changes have been made since I last looked… It is basically a matter of how likely do I think it is going to be that it will conflict with something I am doing. If it is an independent file that is unlikely to conflict with any of my changes, then I'll probably just update and hope for the best."*

The second difference between foraging for change awareness and traditional foraging was in what constituted valued prey. In traditional foraging, predators forage for specific information needed for their task completion. However, in change awareness situations, participants did *not* forage for the latest commits: they could be obtained easily (e.g., via a pull command). Instead, participants foraged to gain a *"partially thorough"* (P1) or less-detailed understanding of the changes in the easily-available commits. In fact, even the *mere presence* (or the *lack*) of new commits could be valuable prey in change awareness.

The third difference between change-awareness foraging and traditional foraging was in how participants balanced costs and benefits. Participants did not expend a lot of effort in knowing about every change in detail. Instead, they adopted a "*light-weight"* approach to foraging: they expended lower costs, to gain less-detailed information. For example, consider the following descriptions of participants' change awareness foraging:

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

> *P9: "Since we have it setup to where each commit that you care about sends you an email, it is nice to just scan the subject lines of the emails… you can kind of see the direction that the code base is going.*

In both the above examples, participants adopted low-cost mechanisms such as *scanning* subject lines (instead of reading) and looking "really quick" (instead of detailed reading). In return, they only gained less-detailed information, instead of a thorough knowledge of the changes.

The fourth difference is in *when* a predator consumes the prey. While a predator forages for a prey and consumes it immediately in traditional foraging situations, the predator might defer prey consumption to later in change-awareness foraging. Predators engaged in a triaging step to decide the necessity to learn about a change immediately, or whether the change awareness could be deferred to later (or never). P12 described his triaging as follows:

> *P12: "I go through email typically twice a day at the beginning and end. 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".*

Such behavior is, indeed, consistent with the findings in Figure 4. We argue that participants considered change awareness "very important" if the change could lead to additional costs (e.g., breaking changes, changes affecting their current task), "important" –but not "very important"– if change awareness was not meant to avoid additional costs, but to ease future foraging (e.g., changes made to a specific code entity of interest) and "not important" when change awareness led to neither benefits nor additional costs. These

above differences in foraging behaviors indicates that change awareness foraging, as a phenomenon, is distinct from traditional information foraging.

However, several aspects of traditional foraging also apply to change awareness foraging. Therefore, bringing an IFT lens to change awareness, as we have done in this section, allows tool builders to leverage IFT's design insights and implementation techniques for change awareness tools. For example, change awareness foraging involves developers learning about interesting changes: this directly corresponds to the category "locate interesting information" or prey-finding patterns in Nabi et al.'s IFT design patterns catalog. Armed with a knowledge of IFT and its design principles and patterns, tool builders can leverage the catalog to evaluate and compare various common design options for their tools (e.g., notifier vs. dashboard for change awareness), thereby speeding up their tool design and implementation.

5.2 Foraging for specific commits

Unlike participants' change awareness foraging that deviated from traditional foraging in several ways, participants' foraging for specific information followed traditional foraging behavior. For example, as participant P3 foraged in his project's version control system to locate a bug, he:

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

These activities reported by P3 map in straightforward ways to the following traditional foraging activities:

- *between-patch foraging*, when the predator decides which among several patches to forage in (e.g., skimming commit messages to decide which commit to go into),
- *within-patch foraging*, when the predator forages *within* a patch (e.g., reading the diff to understand the changes within a commit) and
- *enrichment*, where a predator modifies the environment (e.g., by filtering) to ease the foraging.

Table 2 lists several foraging situations where participants looked for specific information and engaged in traditional foraging behaviors.

Foraging goal	Definition	What patches participants looked in
Understand their own current task progress (others' changes = change awareness)	Look for their own commits for a task, in order to recollect what they did and what remained to be done.	One's own recent commits or commits for a specific task.
Selectively compose changes	Look for specific commits (e.g., bug, feature) to cherry-pick into other branches.	Commits pertaining to the specific bug(s) or feature(s) that need to be cherry-picked.
Understand change impact analysis	Learn about which areas might be impacted by a change, what tests need to be run, etc.	Other commits that modified the code / files / tests that are part of the change.
Debug	Find when a bug was introduced, how the code at that time was, etc.	The bug-introducing commit.
Understand a change / code rationale	Learn why a snippet of code existed or was implemented a certain way.	Commit where the code was added or modified.

Table 2 Foraging for specific information. When participants foraged for specific information, they adopted traditional forms of foraging.

In all these situations where traditional foraging applied, the straightforward mapping from VCS domain to IFT allows us to leverage existing research on IFT-informed design to VCS. In particular, such an approach holds the promise that IFT's design insights and patterns might lend themselves to mitigating the various barriers (Table 3) that participants reported facing in VCS. Table 3 lists the barriers along with the percentage of participants that reported them.

Towards this end, we follow Piorkowski et al.'s lead [36] and draw upon IFT's cost-value proposition: a predator's choices in an environment are an attempt to maximize value per cost. Therefore, a predator's foraging can be improved by improving the costs (C) and values (V) in the environment. However, since the predator's actions are based on their expectations of values E(V) and expected costs E(C), their foraging can also be improved by helping them estimate the values and costs accurately. Piorkowski et al [36] argued that improving programmers' foraging eventually boils down to one of these four ways (listed

in Table 4). In the rest of this section, we demonstrate how these four approaches can be applied to improving VCS environments by addressing the barriers listed in Table 3.

Barrier	Description	% of participants (out of 217)
Non-informative commit messages	Commit messages did not provide enough information about the changes in the commit	66
Tangled changes	Commits contain changes with multiple intents, making it hard to discern the intent behind the changes	54
Information overload	Too many commits in a project, leading to difficulty in searching for information as well as keeping up with the changes	47
Traceability to versions	Ability to trace the entire history of a code snippet, without dealing with fragmentations due to moving code or changing repositories	32
Interpreting diffs	Understanding diffs was hard for large commits and due to noisy changes such as white spaces and line-endings	32
Tool limitations	Limited support for grouping and filtering commits, visualizing the history or for change awareness	20
Traceability to requirement	Trace a change back to its requirements, or locate all changes made for a requirement	20
Traceability to architecture	Trace which components or modules are affected by a change	17

Table 3 Barriers. Participants reported several barriers during their foraging in version control systems.

Aligning Expected with Actual	Align E(V) with V
	Align E(C) with C
Improving Actuals	Increase V
	Decrease C, by decreasing: C_b, between-patch foraging costs, C_w, within-patch foraging costs

Table 4 Four fundamental ways to better support foraging [Piorkowski et al. 2016].

5.2.1 Aligning expected costs & values with actuals

First, consider the first two entries of Table 4, aligning estimates with actuals, so that predators do not make navigation choices based on estimates that are way off from the

actuals. Several participants in our study reported exactly this phenomenon as a barrier, namely an inability to accurately predict costs and/or values.

For example, Table 3 shows that more than 30% of our participants reported difficulties in tracing the entire history of a code snippet. P4 explained one such foraging instance where he looked for when a line "came into being". He looked at the history of the line and navigated to the oldest version there, expecting that was when the line came into being:

P4: "It says 'this line was created in version 721'".

However, he was disappointed: the line was actually moved from elsewhere; therefore, he had to look at the history of where the line was originally located:

P4: "what CVS says is not actually true… let us do a diff between 721 and 720 and we realize: 'yes, this came into being here because it was moved from somewhere else… in version 720 this line corresponds to that line. And the original line came into being at version 507'".

One interpretation of the above example from an IFT's perspective is that of the *actual cost being higher than the expected cost*: P4's disappointment was due to the fact that he expected the oldest version was version 721, but found that history was broken into two segments, namely latest to 721; 720-507. Therefore, at 721, he had to navigate to the older segment of history—an activity involving additional costs than he originally expected.

Participants also made inaccurate estimates of the information value in patches. For example, P9 reported an "*annoying*" instance:

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 due to formatting]."

Here, P9's annoyance was because the actual value in the patch (seeing a section with a lot of changes) was much lower than the value he expected (only one changed line).

One possible reason for predator's inaccurate cost and value estimates is the design of the information environment itself. As P4 noted:

P4: "what CVS tells you is not actually true".

Therefore, one possible direction in which information environments could help predators align their expectations with actuals could be by simply indicating *worst-case* costs and values. For example, in the above instances, simply indicating the worst-case costs (e.g., number of times a line was moved) and values (e.g., number of lines with white-space-only changes) in the environment might help predators better align their expectations of cost and value with the actual cost of getting to a patch and actual value to be found once they get there.

5.2.2 Improving actual costs & values in foraging

The second category of improvements proposed by Piorkowski et al. (Table 4) lies in improving the actuals. This involves reducing actual foraging costs and increasing the actual value of information in patches. Table 5 summarizes how improving the actuals can address some of the barriers that participants reported.

1. *Information overload:* Over 45% of our participants reported facing information overload (Table 3) because they had to sift through too many commits in the VCS. It stands to reason that, in looking for specific information, the majority of these commits were not valuable or relevant to participants. P11 described this situation as:

> *P11: "…there could be some noise from commits I don't care about. Sometimes it is hard to filter out changes that, like I talked about, [were] for the merging and… if they did a big refactor so they renamed a bunch of fields, they are not the person I need to talk to, to understand the context of the particular application that I am trying to look at".*

Here, the "noise" from the irrelevant commits (e.g., merge commits, refactoring commits) diluted the *information value* (i.e., reduced the ratio of relevant commits) in the list of all commits.

Problem	**Problem from IFT perspective**	**Potential Solution (Improving Actuals)**
Information overload	Low information value due to too many commits	Increase V (e.g., filtering pattern)
Traceability to requirement, Traceability to	Information is fragmented and leads to high costs of foraging across different patches or	Decrease C_B (e.g., gather together pattern)

architecture, Traceability to versions	environments (i.e., high between-patch foraging costs, C_B)	
Interpreting diffs	High cost of processing a commit (i.e., high within-patch foraging costs, C_W)	Decrease C_w (e.g., cue decoration pattern to better understand diffs)
Tangled commits		

Table 5 Improving actual costs and values in VCS. Decreasing foraging costs and increasing the information value in the environment can lessen many of participants' barriers in VCS.

2. *Information fragmentation*: Consider the barrier *traceability to requirements* in Table 3: relating a change to its requirements might involve the predator foraging in multiple environments including VCS, bug repository and requirements documents, while locating all changes pertaining to a task might require foraging in multiple commits. Similarly, relating a change to the project's architecture (*traceability to architecture*) might involve foraging in design documents, while looking for a line's entire history or *traceability to versions* might involve the programmer looking at the line's commit history in parts (e.g., P4's example in 5.2.1 involved fragmented history due to code being moved). In all these three traceability-related barriers, the underlying problem is that related information is fragmented: Piorkowski et al. describe this as the "*prey in pieces*" problem. As a consequence, the predator incurs additional *between-patch costs* due to locating and navigating across multiple patches.

3. Interpreting diffs & tangled changes: As Table 3 shows, participants also encountered difficulties while foraging within a commit (within-patch foraging), namely in the commit diffs. First, the only information features for denoting changes in diffs include red and green lines indicating addition or removal of lines: no version control system even distinguishes modified lines. Second, no enrichment mechanisms –such as the ability to filter out white space changes (P9), or grouping all related changes together (e.g., all changes due to a single method rename)—exist within diffs. Third, participants reported issues due to *tangled changes*: commits contained unrelated changes with disparate intents (e.g., refactoring and a bug fix), leaving them with the cognitively expensive option of *mentally* discerning changes and their intents. All these difficulties can be interpreted as high costs of foraging within a commit, or high *within-patch foraging* costs.

Mapping participants' barriers to foraging situations (as shown above), again, allows us to directly apply the corresponding design patterns from Nabi et al.'s catalog [31]. Some examples of applying design patterns to address the barriers in Table 5.

- The *filtering* pattern aims to improve information value in the environment, by eliminating low-value (irrelevant) patches: this can address the issues with information overload.
- The *gather together* pattern can reduce between-patch costs, C_B, by automatically gathering together relevant information from multiple information sources. When applied to VCS, gathering together all commits related to a requirement or gathering together (and chronologically ordering) the disconnected pieces of history for a code snippet can address the barriers relating to traceability to requirements and the traceability to versions.
- The *impact location* pattern also seeks to minimize between-patch costs, C_B, by automatically locating the impact of changing a code location: this is one way to address the traceability to architecture barriers.
- The "reduce cost of processing information" patterns can be applied to improve within-patch costs C_W. For example, combining the *feature tracing* pattern and the *decorator* pattern to decorate the changes with the corresponding features (or intents) can help developers better deal with tangled commits.

5.3 Creating new patches now for foraging in the future

Participants did not simply act as information consumers by foraging in an information environment: they also acted as information producers by changing the environment, in the form of adding new information patches. Examples included splitting uncommitted changes into logical commits (patches), writing commit messages, and notifying other team members (e.g., via emails) of their changes.

Participants recognized that the new patches they were producing would become the information environment for future foragers (consumers). Therefore, as producers, participants often attempted to enable their new patches to support the needs of future

foragers, even if their attempts increased their own costs of creating those patches. Their ways of going about attempting to support future foragers were fundamentally sound: they did so by focusing on decreasing future foraging costs and/or increasing the value of those patches. For example, some participants took care to avoid tangled changes and created small commits containing only a single change, thereby minimizing future foragers' cost of understanding the commit. Some wrote detailed commit messages to offer more cues to future foragers. And some notified other team members of the changes they were making to minimize the team's cost of future change-awareness foraging (recall from 5.1 that participants used emails for change awareness).

In spite of well-intentioned efforts like these, participants as consumers reported difficulties with foraging through the patches they encountered (recall Table 3). In fact, the top two problems in Table 3, namely non-informative commit messages and tangled commits, directly stem from how commit authors created cues (i.e., words in their commit messages) and patches (i.e., commits) to aid future foraging.

One likely factor behind the persistence of these problems lies in the *tension* between different kinds of future foraging needs and between different people. One such example was in the appropriate *granularity* of commits. Several interview and survey participants, such as P6, preferred small commits:

> *P6: "I try to keep all of my commits topical in nature. I try not to have different unrelated changes in the same commit."*

However, he described an instance where he was asked to merge the smaller commits into larger, but fewer commits:

> *P6: "I think I had 7 or 8 commits [as part of the change]. I was asked to merge them down in smaller [number of] commits. It ended up being 2 or 3 commits."*

This was because smaller commits resulted in information fragmentation (discussed earlier in 5.2.2): changes related to the same feature might be split across several commits. Consequently, a developer foraging for this feature has to do so in multiple commits (that are not necessarily contiguous or grouped) and understand the dependencies between them, thereby incurring additional costs.

P6: "[I was asked to merge the commits] because they (commits) are related, they are all adding the same feature. I don't know, I guess some people just had different preferences than I do. I prefer smaller topic commits 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. But then, you get into this dependency graph of changes that is not apparent just by looking through the serial history of commits. Topical commits are not always the best I guess".

P10 also echoed this view and preferred larger commits because a developer can see all corresponding changes without fragmentation, such as during code reviews. He even mentioned the need for a balance between too small vs. too large changes.

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... So we prefer to have checkins be sort of per feature, per requirement, per work item. So there is sort of an art for what's too much. If I am working on a defect, unless the defects are directly related, you should be committing per defect. That is what I typically do. If I am working on a defect and is isolated I'll commit for this defect but if it's different layers, I'll commit all the layers with one commit because it is much easier for us to track that change as one change set for that feature request or bug or whatever."

Participants also reported such tensions regarding the length of commit messages. Some participants preferred detailed commit messages—over 60% of survey participants complained of missing details in commit messages (Table 3). However, others did not always want detailed commit messages, including some participants who preferred shorter ones.

P11: "Commit messages are often read in the command line application so they need to be very short."
Survey participant: "<An ideal commit message> should have variable expectations of length: Large or important changes should have lengthy discussions as necessary; changes warranting no mention ("bug fixes"/"minor changes") should carry no commit message whatsoever and may even be treated as loose amendments to prior commit(s)."

Another kind of tension arose due to conflicting needs for supporting the future foraging needs of *other* programmers (future consumers) versus supporting the present of getting

their task done (present producers). For example, P2 always put his/her own task first by doing extra commits to ease backtracking during his exploratory programming task:

> *P2: "The number 1 motive is returning to code that works. I feel that if I do not have version control I am naked and in any point my software could come crumbling down I would be completely screwed. And it would take hours and hours and hours to get back to something that works. With version control I am never more than 10 minutes away from something that actually works. For me that is the primary motivation / goal for version control".*

However, such temporary commits might add to noise in other future foraging activities—either due to too many commits, or due to code that does not work.

> *P2: "Sometimes defining what was the last version to work is not always an easy question. Just because I committed something that at the time I thought worked might not be necessarily be something that actually works."*

Similarly, P11 also experienced such tensions between meeting the collaboration needs in his current task versus meeting the team's future foraging needs by ensuring a clean commit history where each commit consisted of working code.

> *P11: "I am usually pretty uneasy about what I want to put in a commit because there is (sic) some people that feel that everything that is committed should compile and have running tests all the time. But that kinda limits how often you commit things because then you won't end up committing a day's worth of work in some cases. For a couple of days, I had several broken tests that I didn't have time to go and fix and at some point I had something that wasn't compiling that I needed to share with another developer".*

These problems are difficult, and the IFT lens suggests some possibilities toward removing or at least reducing *some* of them. For example, Nabi et al's patterns such as "*rename*" and "*extract*" aim to ease future foraging by allowing foragers to improve existing patches and cues. Still, the tension situations remain unsolved and IFT –as we have presented it– does not provide any insight into such situations. This is because traditional IFT has only considered a solitary forager, whereas software teams go beyond that.

However, a variant of IFT called *social IFT* goes beyond the solitary forager and deals with foraging in the context of cooperating groups, where the foragers are the individual

members of the group [39]. Social IFT acknowledges the various tensions presented earlier and suggests that there might be a “sweet-spot” to balance the individual-vs-group’s tensions. Software teams are classic examples of such cooperating groups; therefore, we believe that studying VCS through a Social IFT lens can provide us deeper insights into such tensions.

6 DISCUSSION

6.1 Relationship to three-lens model of software history

Participants' foraging in VCS centered around three kinds of activities, namely foraging for change awareness, foraging for change awareness and creating commits that meet future foraging needs. These three activities correspond to the three "lenses" that form the three-lens model of software history proposed in our prior work [11]: the *awareness* lens focuses on change awareness, the *archaeology* lens focuses on seeking specific information and *immediate* lens is for working with current, uncommitted changes to create commits and commit messages. With the idea that developers use different lenses to search for different kinds of information in VCS and that tools need to support activities pertaining to all three lenses, the three-lens model provides a conceptual framework for reasoning about developers' needs in software history and provides the requirements for VCS tools.

Our results provide further insights into developers' foraging behavior in each of the three lenses: pertaining to the *awareness* lens, participants engaged in light-weight foraging; corresponding to the *archaeology lens,* participants mostly engaged in traditional foraging behavior and with the *immediate lens*, participants were mainly concerned with future foraging needs while creating commits. These results uncover the theoretical foundations that underlie the foraging behavior in each of the lenses and enhance the three-lens framework by bringing the principles for the *design* of tools pertaining to the three lenses.

6.2 Open problem: light-weight foraging for change awareness

While making progress towards IFT-informed VCS and software history tools, our results also indicate the limitations of traditional IFT's view of "foraging" as seeking specific, goal-centered information that is necessary for task completion. Prior studies, as well as this thesis, indicate that people also engage in other kinds of foraging activities—particularly, foraging for change awareness. For example, people forage to keep up with the latest news, researchers forage to stay up to-date with the latest developments in their field [43, 14] and developers forage to keep up with the latest changes happening on the

project [13, 11]. Yet, it is surprising how little IFT has focused on people's foraging behavior in change-awareness situations.

Our results suggest that change-awareness foraging is *similar* to traditional foraging in many ways. For example, participants only wanted to know of some "important" –and *not all*– changes happening on the project: this problem of what information to consume is called the "diet problem" in IFT and is addressed by the diet models of foraging [38]. In fact, even the specialized and personalized diet of participants is reminiscent of the findings of Evans et al. and Piorkowski et al. in the web and programming domain [35, 14]. Similarly, Sellen et al. found that, while foraging for news, people heavily relied on headlines and summaries instead of reading entire news articles [43]: this is consistent with IFT's value-cost and scent-following propositions. These similarities suggest that IFT—its constructs, propositions and models—can provide a theoretical framework for studying change-awareness foraging. However, our results (Section 5.1) also indicate several points at which change-awareness foraging is subtly *different* from traditional foraging. First, participants' goal in change-awareness foraging was to minimize future foraging costs rather than task completion. Second, change-awareness foraging was lightweight: participants expended low cost and gained less-detailed information rather than gaining all detailed information; in fact, even the presence or absence of changes might be the prey. Third, during change-awareness foraging, predators deferred prey consumption to later, rather than just-in-time (as and when the changes occurred). These deviations of change-awareness foraging from traditional foraging raises doubts regarding the applicability of traditional IFT in change-awareness situations and calls for in-depth investigation into the question: *how does IFT apply to change-awareness situations?*

6.3 Open problem: Social information foraging in SE

Another situation where traditional IFT falls short is its explanatory power for foraging situations that arise from collaboration. Traditional IFT views foraging as a solitary activity, where the predator works on a task in isolation. While such a "solitary" view closely represents some foraging situations in SE (e.g., individual developer foraging in an IDE), it does not adequately represent other foraging situations that involve collaboration.

For example, traditional IFT has no way of representing the individual *vs.* team tensions described in Section 5.3. Consequently, IFT does not provide much insights into foraging in collaborative situations.

To address this gap, Pirolli proposed a variant of IFT –namely Social IFT– to explain the foraging behavior of individuals who are part of a cooperating group [39]. Software teams are an example of such cooperating groups; therefore, Social IFT can be applied to collaborative SE. Indeed, SE researchers are starting to leverage Social IFT for collaborative situations. For example, recently Bhowmik et al. applied Social IFT to understand how interactions between stakeholders contributed to new requirements in open-source projects [2]. They also used Social IFT to study how team size affects developers' productivity in open-source projects and gained practical insights into optimal team sizes, based on the theoretical findings [3]. These initial successes hold the promise that Social IFT can represent and provide insights into problems concerning collaboration.

This thesis brings to fore new collaborative situations, that offer avenues for research at the intersection of Social IFT and collaborative SE. For example, developers create commits keeping the team's future foraging in mind; however, this leads to different kinds of conflicts between the individual vs. the team. Similarly, we do not know how interactions between developers affects change awareness. We believe research in Social IFT can provide answers to these questions in two ways: first, research needs to focus on operationalizing Social IFT to various situations in SE, given that Social IFT has not yet been widely operationalized. Second, 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.

7 CONCLUSION

Version control systems (VCS) are an important source of information for developers. However, developers face several barriers while seeking information in version control systems. In this thesis, we presented a new perspective of how developers seek information in software supported by VCS: namely the perspective of Information Foraging Theory (IFT).

Among our key results were:

- Participants engaged in traditional foraging behavior while foraging for a specific commit; therefore, VCS tools can leverage existing IFT insights (which are based on traditional foraging) to better support such foraging activities.
- Participants deviated from traditional foraging behavior and adopted *lightweight* (low-cost, low-value) foraging mechanisms to keep up with the latest changes happening on the project; these information needs were also highly personalized. This suggests that change-awareness tools need to allow personalization as well as and low-cost, low-value information-seeking mechanisms, such as personalized instant notifications as well as delayed digests.
- While committing their changes (i.e., creating patches), participants attempted to ease future foraging activities; however, they faced tensions between individual *vs.* team, immediate *vs.* later and between different foraging activities while doing so. VCS tools, therefore, need to support that conflicting requirements with their commits and commit messages.

Our results on participants' foraging also relate in straightforward ways to the conceptual three-lens model of software history that we proposed earlier: participants' change-awareness foraging (Section 5.1) maps to awareness lens, foraging for specific information (Section 5.2) maps to archaeology lens and creating commits (Section 5.3) maps to immediate lens. Our results, therefore, enhance the conceptual three-lens model with concrete insights for designing tools for each of these lenses.

Our results also revealed two major gaps in knowledge at the intersection of IFT and SE. First, participants' light-weight change-awareness foraging was subtly different from traditional foraging; this called for further inquiry into how developers forage to keep up with the latest changes on their project. Second, our results revealed that developers faced different future-foraging related tensions while creating patches (here, commits). These results open up new avenues for research in the cost-value aspects of creating commits, from both traditional as well as social IFT perspectives.

BIBLIOGRAPHY

[1] Alali, Abdulkareem, Huzefa Kagdi, and Jonathan I. Maletic. "What's a typical commit? a characterization of open source software repositories." In *Program Comprehension, 2008. ICPC 2008. The 16th IEEE International Conference on*, pp. 182-191. IEEE, 2008.

[2] Bhowmik, Tanmay, Nan Niu, Prachi Singhania, and Wentao Wang. "On the role of structural holes in requirements identification: an exploratory study on open-source software development." *ACM Transactions on Management Information Systems (TMIS)* 6, no. 3 (2015): 10.

[3] Bhowmik, Tanmay, Nan Niu, Wentao Wang, Jing-Ru C. Cheng, Ling Li, and Xiongfei Cao. "Optimal group size for software change tasks: a social information foraging perspective." *IEEE transactions on cybernetics* 46, no. 8 (2016): 1784-1795.

[4] Biehl, Jacob T., Mary Czerwinski, Greg Smith, and George G. Robertson. "FASTDash: a visual dashboard for fostering awareness in software teams." In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 1313-1322. ACM, 2007.

[5] Brindescu, Caius, Mihai Codoban, Sergii Shmarkatiuk, and Danny Dig. "How do centralized and distributed version control systems impact software changes?." In *Proceedings of the 36th International Conference on Software Engineering*, pp. 322-333. ACM, 2014.

[6] Brun, Yuriy, Reid Holmes, Michael D. Ernst, and David Notkin. "Proactive detection of collaboration conflicts." In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pp. 168-178. ACM, 2011.

[7] Campbell, John L., Charles Quincy, Jordan Osserman, and Ove K. Pedersen. "Coding in-depth semistructured interviews: Problems of unitization and intercoder reliability and agreement." *Sociological Methods & Research* 42, no. 3 (2013): 294-320.

[8] Card, Stuart K., and Jock Mackinlay. "The structure of the information visualization design space." In *Information Visualization, 1997. Proceedings., IEEE Symposium on*, pp. 92-99. IEEE, 1997.

[9] Chi, Ed H., Peter Pirolli, Kim Chen, and James Pitkow. "Using information scent to model user information needs and actions and the Web." In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 490-497. ACM, 2001.

[10] Chi, Ed H., Adam Rosien, Gesara Supattanasiri, Amanda Williams, Christiaan Royer, Celia Chow, Erica Robles, Brinda Dalal, Julie Chen, and Steve Cousins. "The bloodhound project: automating discovery of web usability issues using the InfoScentπ simulator." In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 505-512. ACM, 2003.

[11] Codoban, Mihai, Sruti Srinivasa Ragavan, Danny Dig, and Brian Bailey. "Software history under the lens: a study on why and how developers examine it." In *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*, pp. 1-10. IEEE, 2015.

[12] Cortés-Coy, Luis Fernando, Mario Linares-Vásquez, Jairo Aponte, and Denys Poshyvanyk. "On automatically generating commit messages via summarization of

source code changes." In *Source Code Analysis and Manipulation (SCAM), 2014 IEEE 14th International Working Conference on*, pp. 275-284. IEEE, 2014.

[13] de Souza, Cleidson RB, and David F. Redmiles. "An empirical study of software developers' management of dependencies and changes." In *Proceedings of the 30th international conference on Software engineering*, pp. 241-250. ACM, 2008.

[14] Evans, Brynn, and Stuart Card. "Augmented information assimilation: Social and algorithmic web aids for the information long tail." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 989-998. ACM, 2008.

[15] Fleming, Scott D., Chris Scaffidi, David Piorkowski, Margaret Burnett, Rachel Bellamy, Joseph Lawrance, and Irwin Kwan. "An information foraging theory perspective on tools for debugging, refactoring, and reuse tasks." *ACM Transactions on Software Engineering and Methodology (TOSEM)* 22, no. 2 (2013): 14.

[16] Gutwin, Carl, Reagan Penner, and Kevin Schneider. "Group awareness in distributed software development." In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pp. 72-81. ACM, 2004.

[17] Gutwin, Carl, Kevin Schneider, David Paquette, and Reagan Penner. "Supporting group awareness in distributed software development." In *International Workshop on Design, Specification, and Verification of Interactive Systems*, pp. 383-397. Springer, Berlin, Heidelberg, 2004.

[18] Guzzi, Anja, Alberto Bacchelli, Yann Riche, and Arie van Deursen. "Supporting developers' coordination in the ide." In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, pp. 518-532. ACM, 2015.

[19] Henley, Austin Z., Alka Singh, Scott D. Fleming, and Maria V. Luong. "Helping programmers navigate code faster with Patchworks: A simulation study." In *Visual Languages and Human-Centric Computing (VL/HCC), 2014 IEEE Symposium on*, pp. 77-80. IEEE, 2014.

[20] Henley, Austin Z., Scott D. Fleming, and Maria V. Luong. "Toward Principles for the Design of Navigation Affordances in Code Editors: An Empirical Investigation." In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pp. 5690-5702. ACM, 2017.

[21] Kasi, Bakhtiar Khan, and Anita Sarma. "Cassandra: Proactive conflict minimization through optimized task scheduling." In *Proceedings of the 2013 International Conference on Software Engineering*, pp. 732-741. IEEE Press, 2013.

[22] Kawrykow, David, and Martin P. Robillard. "Non-essential changes in version histories." In *Proceedings of the 33rd International Conference on Software Engineering*, pp. 351-360. ACM, 2011.

[23] Kim, Sunghun, E. James Whitehead Jr, and Yi Zhang. "Classifying software changes: Clean or buggy?." *IEEE Transactions on Software Engineering* 34, no. 2 (2008): 181-196.

[24] Ko, Andrew J., Brad A. Myers, Michael J. Coblenz, and Htet Htet Aung. "An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks." *IEEE Transactions on software engineering* 32, no. 12 (2006).

[22] Ko, Andrew J., Robert DeLine, and Gina Venolia. "Information needs in collocated software development teams." In *Software Engineering, 2007. ICSE 2007. 29th International Conference on*, pp. 344-353. IEEE, 2007.

[26] Kuttal, Sandeep Kaur, Anita Sarma, and Gregg Rothermel. "Predator behavior in the wild web world of bugs: An information foraging theory perspective." In *Visual Languages and Human-Centric Computing (VL/HCC), 2013 IEEE Symposium on*, pp. 59-66. IEEE, 2013.

[27] Lawrance, Joseph, Rachel Bellamy, and Margaret Burnett. "Scents in programs: Does information foraging theory apply to program maintenance?." In *Visual Languages and Human-Centric Computing, 2007. VL/HCC 2007. IEEE Symposium on*, pp. 15-22. IEEE, 2007.

[28] Lawrance, Joseph, Rachel Bellamy, Margaret Bumett, and Kyle Rector. "Can information foraging pick the fix? A field study." In *Visual Languages and Human-Centric Computing, 2008. VL/HCC 2008. IEEE Symposium on*, pp. 57-64. IEEE, 2008.

[29] Lawrance, Joseph, Margaret Burnett, Rachel Bellamy, Christopher Bogart, and Calvin Swart. "Reactive information foraging for evolving goals." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 25-34. ACM, 2010.

[30] Marzban, Maryam, Zahra Khoshmanesh, and Ashkan Sami. "Cohesion between size of commit and type of commit." In *Computer Science and Convergence*, pp. 231-239. Springer, Dordrecht, 2012.

[31] Nabi, Tahmid, Kyle MD Sweeney, Sam Lichlyter, David Piorkowski, Chris Scaffidi, Margaret Burnett, and Scott D. Fleming. "Putting information foraging theory to work: Community-based design patterns for programming tools." In *Visual Languages and Human-Centric Computing (VL/HCC), 2016 IEEE Symposium on*, pp. 129-133. IEEE, 2016.

[32] Nguyen, Anh Tuan, Michael Hilton, Mihai Codoban, Hoan Anh Nguyen, Lily Mast, Eli Rademacher, Tien N. Nguyen, and Danny Dig. "API code recommendation using statistical learning from fine-grained changes." In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 511-522. ACM, 2016.

[33] Niu, Nan, Anas Mahmoud, Zhangji Chen, and Gary Bradshaw. "Departures from optimality: understanding human analyst's information foraging in assisted requirements tracing." In *Proceedings of the 2013 International Conference on Software Engineering*, pp. 572-581. IEEE Press, 2013.

[34] Piorkowski, David, Scott Fleming, Christopher Scaffidi, Christopher Bogart, Margaret Burnett, Bonnie John, Rachel Bellamy, and Calvin Swart. "Reactive information foraging: An empirical investigation of theory-based recommender systems for programmers." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1471-1480. ACM, 2012.

[35] Piorkowski, David J., Scott D. Fleming, Irwin Kwan, Margaret M. Burnett, Christopher Scaffidi, Rachel KE Bellamy, and Joshua Jordahl. "The whats and hows of programmers' foraging diets." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 3063-3072. ACM, 2013.

[36] Piorkowski, David, Austin Z. Henley, Tahmid Nabi, Scott D. Fleming, Christopher Scaffidi, and Margaret Burnett. "Foraging and navigations, fundamentally: developers'

predictions of value and cost." In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 97-108. ACM, 2016.

[37] Pirolli, Peter, and Stuart Card. "Information foraging in information access environments." In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 51-58. ACM Press/Addison-Wesley Publishing Co., 1995.

[38] Pirolli, Peter, and Stuart Card. "Information foraging." *Psychological review* 106, no. 4 (1999): 643.

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

[40] Ragavan, Sruti Srinivasa, Sandeep Kaur Kuttal, Charles Hill, Anita Sarma, David Piorkowski, and Margaret Burnett. "Foraging among an overabundance of similar variants." In Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, pp. 3509-3521. ACM, 2016.

[41] Ragavan, Sruti Srinivasa, Bhargav Pandya, David Piorkowski, Charles Hill, Sandeep Kaur Kuttal, Anita Sarma, and Margaret Burnett. "PFIS-V: Modeling Foraging Behavior in the Presence of Variants." In Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, pp. 6232-6244. ACM, 2017.

[42] Sarma, Anita, David F. Redmiles, and Andre Van Der Hoek. "Palantir: Early detection of development conflicts arising from parallel code changes." *IEEE Transactions on Software Engineering* 38, no. 4 (2012): 889-908.

[43] Sellen, Abigail J., Rachel Murphy, and Kate L. Shaw. "How knowledge workers use the web." In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 230-234. ACM, 2002.

[44] Shivaji, Shivkumar, E. James Whitehead, Ram Akella, and Sunghun Kim. "Reducing features to improve code change-based bug prediction." *IEEE Transactions on Software Engineering* 39, no. 4 (2013): 552-569.

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

[46] Sillito, Jonathan, Gail C. Murphy, and Kris De Volder. "Questions programmers ask during software evolution tasks." In *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, pp. 23-34. ACM, 2006.

[47] Spool, Jared M., Christine Perfetti, and David 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).

[48] Tao, Yida, Yingnong Dang, Tao Xie, Dongmei Zhang, and Sunghun Kim. "How do software engineers understand code changes? : an exploratory study in industry." In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, p. 51. ACM, 2012.

[49] Zanjani, Motahareh Bahrami, George Swartzendruber, and Huzefa Kagdi. "Impact analysis of change requests on source code based on interaction and commit histories." In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pp. 162-171. ACM, 2014.

[50] https://web.engr.oregonstate.edu/~srinivas/software-history.html