Journal of Computer Languages 62 (2021) 101010

Contents lists available at ScienceDirect

Journal of Computer Languages

journal homepage: www.elsevier.com/locate/cola

How end-user programmers forage in online repositories? An information foraging perspective

Sandeep Kaur Kuttal *, Se Yeon Kim, Carlos Martos, Alexandra Bejarano

Tandy School of Computer Science, University of Tulsa, Tulsa, OK, USA

ARTICLE INFO

Keywords:
Variants
Online repositories
Information Foraging Theory
User studies

ABSTRACT

End-user (non-professional) programmers often opportunistically create programs, they evaluate various alternatives and reuse existing code by merging components from it or modifying it to suit the context or problems of their programs. Finding and evaluating which program variants to reuse code from is challenging because the searching mechanisms within online repositories are not optimal. To understand the reuse behavior of end-user programmers and to provide implications on how to further support them, we conducted an empirical study in which eight end-user programmers foraged in online repositories, specifically App Inventor Gallery and File Exchange. Using Information Foraging Theory, we qualitatively analyzed the end-user programmers' behavior and focused on not only program variants from a single source, but also on similar variants from various sources developed over time and by different authors. This analysis revealed new cue types and strategies specific to novice and experienced end-user programmers as they foraged between- and within-variants.

1. Introduction

Programming is a creative task and is generally exploratory in nature, allowing programmers to opportunistically create programs through code reuse by "gluing" together components or modifying existing code to suit a new context or problem [1,2]. This behavior is often seen in end-user (non-professional) programmers (EUPs) as they use trial-and-error to create code opportunistically or incrementally, and rarely follow the standard software lifecycle [3]. One way to enable this behavior of code reuse is through online repositiories.

Online repositories contain programs and their variants created by EUPs e.g., File Exchange [4] for MATLAB [5], App Inventor Gallery [6] for App Inventor [7], and Scratch repository [8] for Scratch [9]. Past research has shown that 43% of programs submitted to an online repository were variations of previously submitted programs [10]. These code variants are created because of the provision of creating program clones or allowing the use of subprograms [11]. Although EUPs may view their programs as "throw-away", their code is often long-lived and, in many cases, is reused by other EUPs [12,13]. However, within an online repository, finding and reusing appropriate variants of a program is a challenging task.

The challenge lies with EUPs' information seeking behavior, which requires a higher cognitive effort for EUPs and may not be fully supported by the tools provided within online repositories. During information seeking for a reuse task, EUPs (1) forage and differentiate between contextually similar variants, (2) localize the appropriate variant, and (3) evaluate the context compatibility of the selected variant's code snippets to the desired program variant. Furthermore, the information-seeking behavior differs between novice and experienced programmers [14]. To understand the information seeking behavior of both novice and experienced EUPs, we posit that Information Foraging Theory (IFT) – a theory on information-seeking behavior – can help us understand how EUPs forage for reusable code among program variants.

In IFT, a predator (EUPs) forages for prey (e.g., program variant or code snippet) by following cues (e.g., labels on links) in patches (e.g., web pages, IDEs). IFT has been studied and applied in connection with the process of "foraging" by web users [15–18] and navigating by professional programmers while debugging programs [19–25]. Additionally, IFT has been used to study the debugging behavior of EUPs [26,27] and the reuse behavior of programmers when foraging program variants from the same project [28,29]. However, IFT is unexplored in the domain of EUPs reusing code in online repositories.

In this paper, we explore the utility of IFT to understand the reuse behavior of novice and experienced EUPs in online repositories. We conducted an empirical study of novice and experienced EUPs and qualitatively analyzed their information-seeking behavior. Furthermore, this paper is an extension of our shorter research paper [30]

* Corresponding author.
E-mail addresses: sandeep-kuttal@utulsa.edu (S.K. Kuttal), seyeon-kim@utulsa.edu (S.Y. Kim), carlos-martos@utulsa.edu (C. Martos), alb146@utulsa.edu (A. Bejarano).

https://doi.org/10.1016/j.cola.2020.101010
Received 27 June 2020; Received in revised form 26 September 2020; Accepted 1 November 2020
Available online 9 November 2020

and presents (1) a new research question (RQ2) regarding integration of variants, (2) further analysis of the study transcripts introducing new cue types and strategies, (3) a comparison of novice and experienced EUPs, and (4) implications for IFT theory. The primary research questions we investigated were:

RQ1: What are the *types of information* which help EUPs identify variants that can be reused?

- *Between-variant foraging*: How do EUPs forage between variants of an artifact to find and evaluate a potential variant?
- *Within-variant foraging*: How do EUPs forage within a specific variant to find and evaluate a potential patch?

RQ2: How do EUPs *integrate* parts of the program across variants?

This paper is structured as follows: Section 2 describes the background on variations and IFT, Section 3 describes our applied methodology, Section 4 presents our results organized in accordance with the reuse model of variants, Section 5 discusses the timeline of participants' behavior and compares cues used by novice and experienced EUPs, Section 6 provides implications for theory and tools, Section 7 describes threats to validity, Section 8 presents related works affiliated with code reuse in online repositories and IFT in software engineering, and Section 9 consists of our conclusion.

2. Background

In this section, we discuss the background on variations and information foraging theory.

2.1. Variations

Variations are "when multiple related implementations exist either serially or in parallel" and variants are "syntactically valid program[s] that occurs together with similar, related programs in a group" as defined by Ragavan et al. [28]. Users can access and modify the contents of these variants to create a newly modified program according to their desired specifications. For example, in File Exchange, a MATLAB seven-segment display project has several variations that were created over time and by different developers. The original project and each variation of that project is referred to as a variant that can be further reused. These variants are often stored by EUPs and made available to other EUPs within online repositories.

2.2. Information foraging theory

Pirolli and Card derived Information Foraging Theory (IFT) from optimal foraging theory [31,32]. IFT models how EUPs (predators) follow scents emanated from cues to track program variants (prey). Cues are information features, such as labels of web links, which give users direction of where to go. These elements are found in information sources which, in the context of this experiment, include variants and patches (such as the programming environment, file explorers, and web pages). Predators optimize their foraging behavior constructs based on the perceptions/estimates of the value and cost of a patch, which determines their navigations; according to IFT, this is known as scent construct [32]. Predators also modify their environment for enrichment [32,33].

IFT has been used to examine and explain the behavior of users as they sift through information on the web [15–18] and for developing models to predict user navigations through the web [15,17]. These IFT models predict user behavior in close precision to the strength of a scent trail in a web environment.

3. Methodology

To understand the reuse behavior of novice and experienced EUPs in online repositories, we conducted further analysis of our previous empirical study of EUPs reusing program variants in two different environments that are popular among EUP communities, App Inventor Gallery and File Exchange [30]. In this study, participants were given two information-seeking tasks that allowed us to observe cues and strategies used while foraging between- and within-variants. In this section, we describe our applied methodology for the study.

3.1. Environment

3.1.1. App Inventor Gallery (AIG)

App Inventor Gallery is an online repository of App Inventor applications. The App Inventor environment is a software development platform designed for mobile Android programs. We specifically chose App Inventor Gallery for our study because: (1) As of 2020, the App Inventor community has 8.2 million registered users from 195 countries and about 230,000 active weekly users that in all have used App Inventor to build 34 million apps [7]. (2) App Inventor is popular with the sub-community of EUPs such as formal and informal educators, government and civic employees and volunteers, designers and product managers, hobbyists and entrepreneurs, and researchers [7]; which make up our participant selection pool. (3) App Inventor is a visual language that is easy to use and allows users of varied experiences to create apps. (4) App Inventor allows users to share their apps with other users.

The arrows in Fig. 1 show transitions of a user's foraging in AIG. The user begins searching (enrichment) for variants in a search bar using keywords and a list of relevant variants are returned.

Opening a variant exhibits to the user three patches namely, the designer, the code editor, and the emulator (from left to right in Fig. 1). The designer patch allows the user to design the interface of a mobile app, the code editor allows implementation of the app using blocks, and the emulator displays the output (after execution) of the created app.

3.1.2. File Exchange (FE)

File Exchange is an online repository of MATLAB applications. We specifically chose the File Exchange repository for our study because: (1) MATLAB is a popular scientific and engineering programming language for EUPs at universities and companies [34]. (2) Our university was actively teaching and using MATLAB, making it easier to find participants with experience in MATLAB. (3) MATLAB allows users to share their code with other users.

Similar to AIG, arrows in Fig. 2 indicate transitions of a user foraging in FE with a noted difference that a list of patches, in which a user can select, will be returned upon variant selection. With the keyword-based search, a user is able to find list of variants. On selecting a variant, a user can download a list of patches (files) and then open the individual patches in the MATLAB IDE.

3.2. Participants

Since EUPs are programmers who have no formal professional software engineering training, we recruited eight non-Computer Science students from the University of Tulsa. We recruited these students by sending emails and posting flyers within the Engineering and Natural Sciences department as many of the students within that department have some programming experience. Students then voluntarily chose to respond to our postings and participate in our study and were compensated with a $20 gift card. Participants were selected after filling out a background questionnaire to screen for EUPs. The demographics of the participants are shown in Table 1.

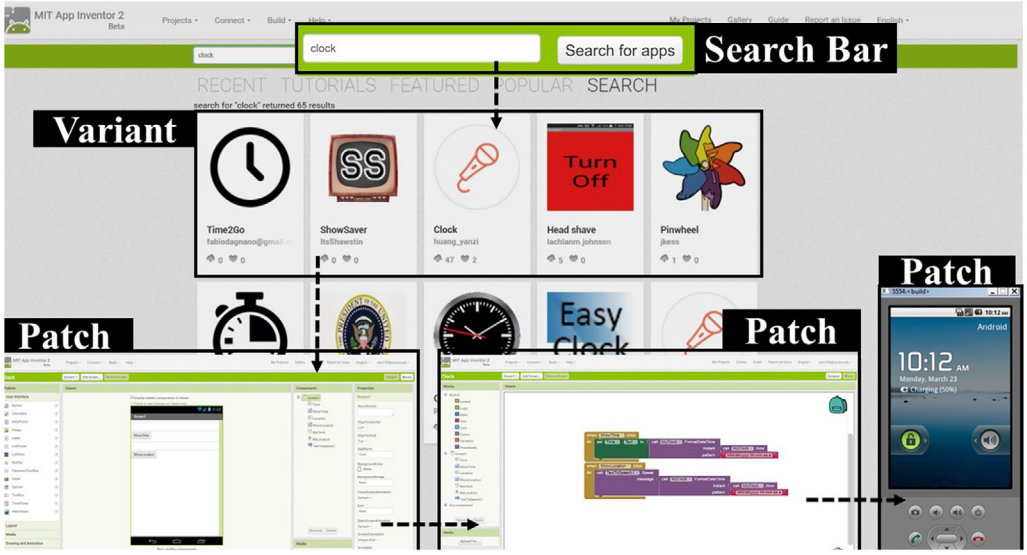

Fig. 1. App Inventor Repository with variants and patches.

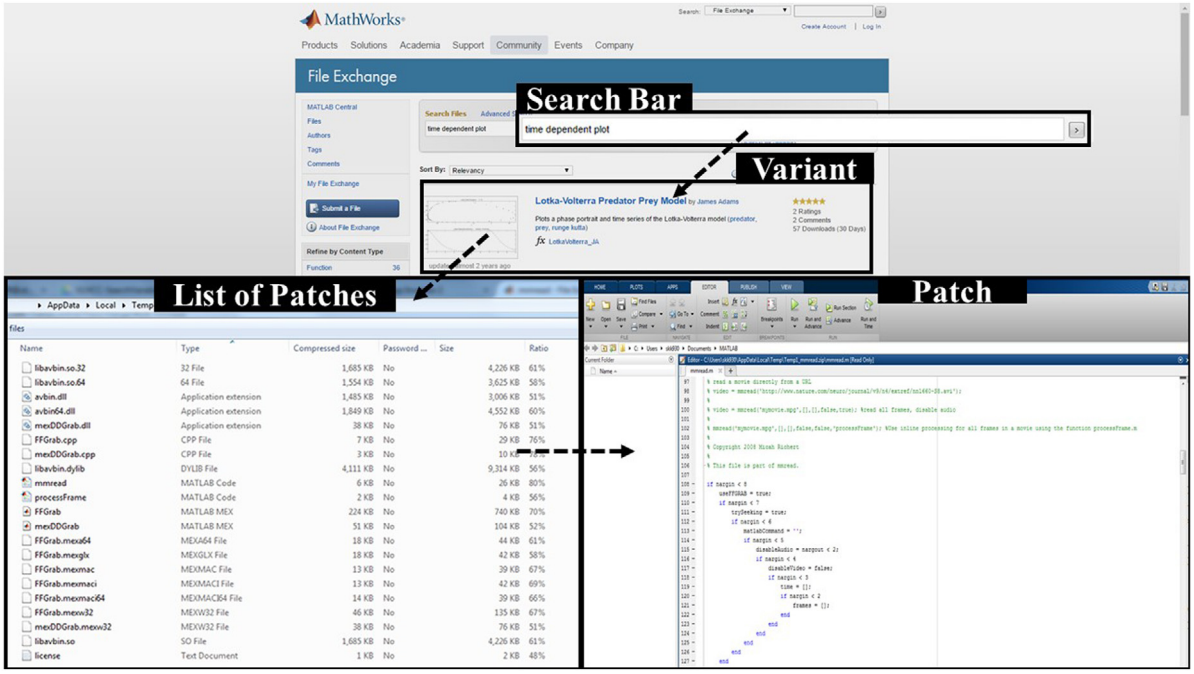

Fig. 2. File Exchange Repository with variants and patches.

Table 1
Participant Demographics.

Participant	Age	Gender	Education level	Major	Programming experience in	Programming experience
P1	19–23	Female	Sophomore	Mechanical Engineering	App Inventor - None	Less than a year
P2	30–40	Male	PhD or similar	Mechanical Engineering	App Inventor - None	Greater than 4 years
P3	19–23	Male	Junior	MIS	App Inventor - None	3 years
P4	24–29	Male	Masters	Geophysics	App Inventor - None	2 years
P5	24–29	Female	Masters	Geophysics	MATLAB - Greater than 4 year	Greater than 4 years
P6	30–40	Male	PhD or similar	Mechanical Engineering	MATLAB - Greater than 4 years	Greater than 4 years
P7	24–29	Male	Masters	Chemical Engineering	MATLAB - 2 years	Less than a year
P8	19–23	Male	Masters	Electrical Engineering	MATLAB - 2 years	2 years

To qualitatively observe participants' foraging behaviors in-depth, we used a small but more generalizable population by dividing the participants into two groups: App Inventor Gallery (AIG) and File Exchange (FE). We used AIG and FE participants to categorize novice and experienced end-user programmers respectively and to understand how experience can be a factor in information-seeking behaviors [14]. While all participants had some experience with programming, AIG participants were novice as they had no experience programming in

App Inventor. Conversely, FE participants were experienced MATLAB programmers as they had 2 or more years of experience programming in MATLAB as shown in Table 1.

3.3. Study design

Prior to the study, participants were asked to fill out a background questionnaire. AIG participants completed an additional short tutorial on App Inventor. We then conducted a formative lab study on the participants using think-aloud protocols [35] asking participants to vocalize their thoughts and feelings when performing their tasks. Participants were given 50 min to complete two tasks. During the study, a lead researcher observed participants within the same room and the participants' verbalizations and on-screen actions were recorded using screen-capture software Morae [36] (signed consent from participants was acquired beforehand). Thus, we hoped to gain insights into participants' thought processes and barriers they faced while exploring, understanding, and selecting between- and within-variants.

The study was followed by a retrospective interview conducted by the lead researcher who had been observing the tasks. We used interviews to elicit participants' experiences and knowledge, while the participants foraged for variants and patches. During the interview, we played back recordings of their actions and verbalizations to seek insights into participants' behavior. Furthermore, the interviews helped us triangulate hence, increased the generalizability of our lab findings [37]. The retrospective interview results were also audio recorded with screen capture. Both the study and retrospective interview required us to administer the study participants on an individual basis with an observer.

3.4. Task context

Two tasks were chosen to observe cues and strategies that participants used while foraging between- and within-variants. To facilitate between-variant foraging behavior, we selected tasks based on two factors: popularity of projects and late-appearance of variant in the search results to enable more foraging. The tasks were designed such that by using common keywords, the intended project would appear on the second or third page of the query results. Task 1 aimed to find a project (destination variant) to integrate code into. Task 2 aimed to find another project (source variant) to integrate code from. While the tasks had multiple solutions, the ideal solution for Task 1 was to find a project that needed no modification. The ideal solution for Task 2 was to find and integrate part(s) of the code of a project into the previous project found in Task 1.

Task 1: The first task was designed to observe between- and within-variant foraging.

AIG participants were given a scenario of Bob who was visually impaired and were instructed to find a project (variant) for Bob with the specifications to "speak aloud the current date and time when he manually selects a button 'Time''.' AIG participants were given a task with this scenario as it had a realistic and imaginative context [38] to motivate and maximize the engagement of these novice EUPs.

Meanwhile, since all *FE participants* were experienced MATLAB programmers, they were not given a scenario. Instead, they were told that "[while] working with signals or fields that vary in time, it is often useful to visualize those fields using animations" and were asked to find a project (variant) that had a time-varying graph which allows users "to input a 1- or 2-dimensional time-dependent variable" and "the option to save results as a video file for later playback when MATLAB isn't available".

Task 2: Once the participants felt they finished Task 1, they moved onto Task 2. The second task was designed for between- and within-variant foraging as well as integration by having participants, ideally, find another project to integrate into the project found in Task 1.

AIG participants continued the scenario from Task 1 and were asked to change the previous program to allow "voice commands instead of manual selection" because Bob had "difficulty accessing the button widget".

Meanwhile, *FE participants* were asked "to use the videos [from Task 1] to extract differences between two plots" because the "version of MATLAB [software] has changed".

3.5. Qualitative analysis

To correctly identify cues associated with participants' foraging behaviors, we qualitatively coded the data by first using cues established by previous works [28,39]. For behaviors/cues not described by previous works, we created new cues to code the phenomena exhibited by participants in the study. We segmented the transcripts of participants' think-aloud videos into 30-second segments (multiple cues can be in one segment). To mitigate for possibly overlooked or incorrectly defined cues/behaviors, two researchers coded the transcripts of participants' actions. We then used the Jaccard's measure to calculate inter-rater reliability [40] and achieved 89% agreement. After an agreement was made, one researcher coded the rest of the transcripts of participants' verbalization and actions. The code set used for the study can be found in Table 2.

4. Results

To demonstrate the resulting behavior of our participants while forging we made use of the reuse model from previous research [28, 41]. Rosson & Carroll modeled programmers' reuse of a single variant of source code [41]. The programmers find and implement code from one "usage context" (reusable code) to complete their current task "current context" (integrating the reusable code). Ragavan et al. [28] extended the model to accommodate multiple variants. Fig. 3 outlines the Reuse Model of variants into three major stages: (1) finding and evaluating the current context, (2) finding and evaluating the usage context, and (3) the integration [28]. Programmers can return (backtrack) to variants or patches at any time while foraging to reuse variants/patches. While Ragavan et al. focused on variants developed from a single source over time, we focused on similar variants from various sources developed over time (temporally) and space (spatially i.e. by different authors). We also modified the reuse model to include Overall Strategies.

In this section, we discuss the overall strategies (Section 4.1) used throughout the reuse model (Fig. 3). Table 3 provides the definitions for each component of the reuse model and the corresponding RQs. We address RQ1 by investigating the current context and usage context of variants and patches (Sections 4.2 and 4.3) to determine the types of information that help EUPs identify variants that can be reused. Specifically, RQ1 (a) was investigated by observing the participants' behavior between-variant foraging for variants while they find a destination variant (Section 4.2.1) and then evaluate it (Section 4.2.2); similarly, while they find a source variant and evaluate it (Section 4.3.1). RQ1 (b), was investigated by observing participants' behavior within-variant foraging for patches while they find a destination patch (Section 4.2.3) and then evaluate it (Section 4.2.4); similarly, while they find a source patch and evaluate it (Section 4.3.2). We also address RQ2 by examining the integration of program code across variants (Section 4.4) to determine how EUPs integrate parts of a program across variants.

4.1. Overall strategies

Bridging Gaps in Knowledge: Participants gained information and continuously reassessed their knowledge base for what they did or did not know (gaps in knowledge [42]) while foraging between- and within-variants, which changed the participants' goal or strategy. We call this behavior *Bridging Gaps in Knowledge for Strategy* and *Goal*, respectively.

Table 2
This codeset describes the cue-types, operations, navigation and enrichment associated with participants' actions.

Code	Description
Cue-Types	
Visual	Cues based on graphics
Text: Filename	Filename cues
Text: Description	Description-inspired cues
Author	Author-name-inspired cues
Quality	The amount of functions a variant performs
Rating	Cues based on the rating of a variant
Timestamps	Dates which mark variants e.g., last update, creation date
Familiarity	Cues based on how a user recognizes the patch or variant
Output	Output-inspired cues
Code	Cues from the source code, e.g., function names, block labels
Documentation: Internal	Document-inspired cues native to the repositories and their contents
Documentation: External	Document-inspired cues from other sources outside the repositories
Code status	Cues which describe or display errors in the code non-intrusively
Operation	
Comparison: Variant	Compare two variants
Comparison: Patches	Compare two patches
Navigations	
Between Variant Navigation	All cues which guided between-variant navigation
Between Patch Navigation	All cues which guided within-variant navigation
Enrichment	
Internal Search	Marked with cues relating to internal search engines (FE, AIG)
External Search	Marked with cues relating to external search engines (Google, Yahoo!)
Between Variant Enrichment	All other cues which guided enrichment

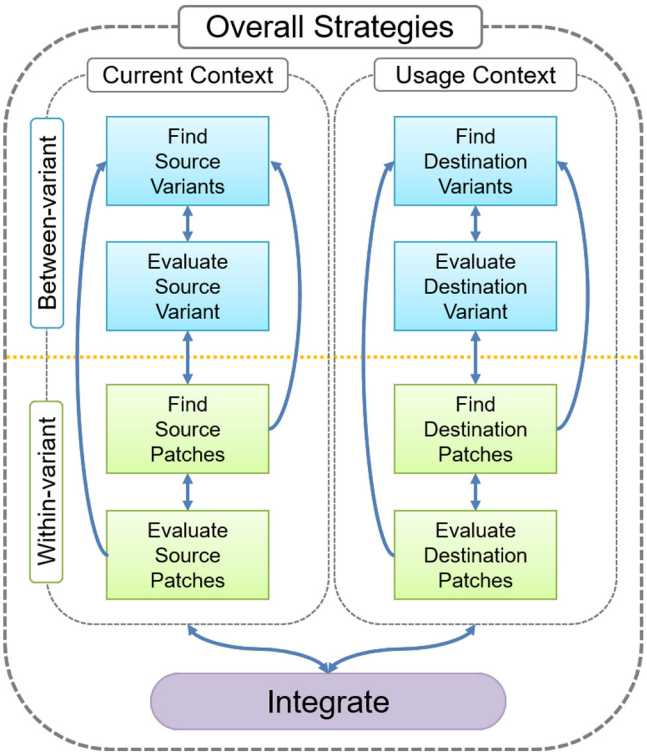

Fig. 3. Reuse model.
Source: Adapted from Ragavan [28].

When participants' gaps in knowledge changed their strategy, we characterized it as *Bridging Gaps in Knowledge for Strategy*. For example, P1 set a goal to find a variant that satisfied Task 1 and assessed her knowledge then used keywords from Task 1 in her search criteria (*Keywords From the Task* an enrichment strategy later mentioned in this study) to find the variant. However, after using the strategy, P1 reassessed her knowledge base with the results of her enrichment strategy and realized a gap in knowledge that her strategy was inefficient as she commented "*it is too general.*" To bridge the gap of knowledge for strategy, she changed to a navigational strategy by using *Recency* and commented she "*[thought] it was one of the more popular apps or more recently chosen.*" However, after not seeing the results she wanted, P1 went back to an enrichment strategy, this time *Filtering Search Results* by using quotation marks (a Google filter shortcut) commenting "*that [quotation marks to filter] kind of tricks would work here [in AIG]*".

Table 3
Reuse model components and their definitions.

	Reuse model components	Definitions
	Contexts	
	Current context	The source which contains reusable code
	Usage context	Destination, the location where reusable code needs to be integrated
	Overall strategies	
	Bridge gaps in Knowledge	Users changed their goals and strategies based on the information gained
	Comparing source & Destination	Users compared source and destination variant/patch to evaluate source variant/patch
RQ1	**Between-variants**	
	Find destination variant	Identify the variant base for future integration
	Evaluate destination variant	Determine whether the variant base is within the context of the current task
	Find source variant	Identify the variant that contains reusable code
	Evaluate source variant	Determine whether the selected variant is usable within the context of the current task
RQ2	**Within-variants**	
	Find destination patch	Identify the patch base for future integration
	Evaluate destination patch	Determine whether the patch base is within the context of the current task
	Find source patch	Identify the patch that contains reusable code
	Evaluate source patch	Determine whether the selected patch is usable within the context of the current task
RQ3	Integrate	Integrate reusable code into variant/patch base

When participants' gaps in knowledge changed their goal, we call this *Bridging Gaps in Knowledge for Goal.* This is evident as P7 kept the same strategy but changed his goal after assessing his knowledge base for gaps. P7's goal was to find a variant (project) that satisfied Task 1. However, after using enrichment strategies in File Exchange, P7 reassessed his knowledge base and realized his search results only returned plotting graphs. He noted the gap in knowledge as he mentioned "*could it be two different downloads that I'm looking for?*" Thus, to bridge the gap in knowledge for his goal, P7 changed his goal from finding one app to finding two apps and continued using an enrichment strategy to find a variant (project) that can "*take to media control... to record*". This is similar to the behavior found by Lawrance et al. [23] in which the goals of their participants changed while debugging .

Comparing Source and Destination: Comparing source and destination variants/patches occurred anytime the participants evaluated a source variant/patch. FE participants did compare between source and destination patches, while AIG participants did not since they persisted with the destination variant/patch they chose and did not search for different variants/patches.

After FE participants obtained the desired source patch, they had three choices to make: (1) keep the destination variant/patch, (2) switch between the source and destination variant/patch, (3) or abandon either the source or destination variant/patch. To make these three choices, FE participants compared the variants/patches either mentally in their head or physically on screen. For example, after evaluating a destination variant mentally, P8 compared the destination variant and source patch and then stated "*this is exactly what I am looking for*" and moved on to Task 2. Other participants compared variants physically such as when P5 compared combinations of variants and patches (see Fig. 4) next to each other. While choosing between the three choices, FE participants reevaluated the destination patch and better understood the source patch through comparison.

4.2. Stage 1: Finding and evaluating current context

In the current context, a predator (EUP) forages between- and within-variants to find and evaluate variants/patches to establish a destination variant/patch and aid their current task.

4.2.1. Find destination variants

In the current context, while finding a destination variant to use as a base for future integration, our participants searched for variants in online repositories using internal or external search engines.

Internal Search: Any searches executed in domain-specific online repositories (e.g., searching code in AIG or in FE) are referred to as internal searches. We found AIG participants made more searches in the site's search engine compared to FE participants. With the exception of P1, all of the AIG participants searched internally. AIG participants made 93% of queries in the AIG search engine versus FE participants who made 62% of queries in the FE search engine. This suggests that AIG participants preferred using internal search engines as these may give more domain-specific search results. Similarly, FE participants switched to the internal search engine for similar reasons. P7 said, "*I wanted to narrow my search a little more to MATLAB-specific stuff*". These results align with previous research, which found that participants preferred internal search engines when their goals were defined and external search engines when they wanted more generic results [43].

External Search: Any searches made in generic search engines (e.g., searches in Google or Bing) are referred to as external searches. Only one AIG participant and three FE participants made external searches. While all participants made internal searches, only half made external searches. We observed that our participants expected internal search engines to behave as external search engines. For example, P1 searched in the Google Search engine for "how to search using keywords" and used those techniques to search in AIG. Although these techniques were not helpful, this behavior indicates that users may expect internal search engines to behave like external search engines.

Enrichment Strategies: Participants made use of enrichment strategies during internal and external searching. An enrichment strategy is when a predator "mold[s] the environment" [33] to enrich the information. As similarly described by Teevan et al. participants used the "Orienteering" approach while formulating queries [44]. While both AIG and FE participants utilized keywords from the task for their first query formulation, only FE participants used the following methods:

Domain Keywords: Some participants had experience using online repositories and used more domain-specific keywords. For example, P7

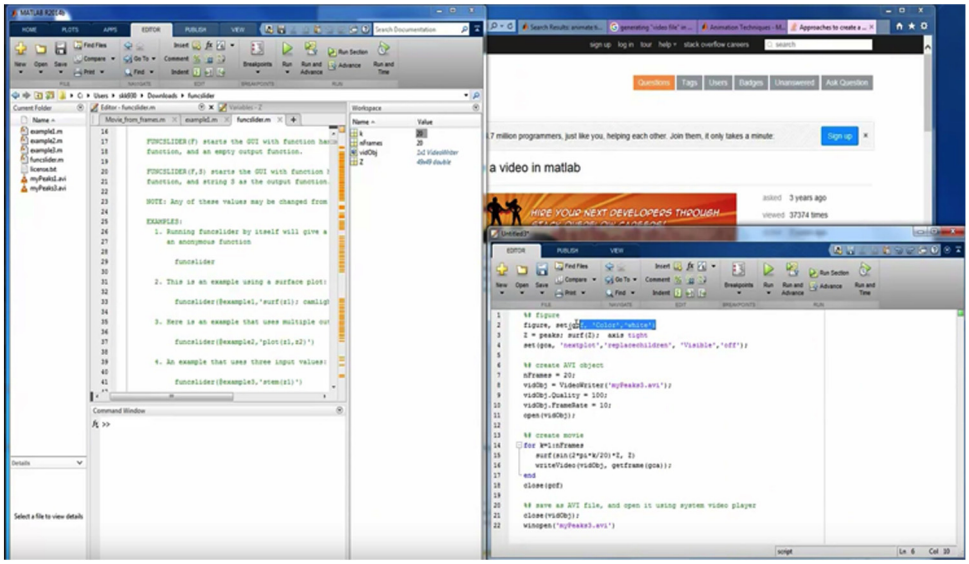

Fig. 4. P5 physically compares source (left) and destination (right) patches.

used the keyword "plot" instead of "graph" as he commented "because I feel like MATLAB uses plots".

Semantically Similar Keywords: When searching for variants using keywords, participants saw the recurrence of specific word(s) even with new query formations. They used these reoccurring words to reformulate their queries to get better search results. For example, P5 commented "*I am writing 'video files.' that's not how MATLAB called [it],* they used the word 'animate,' so I discovered later that I need use the word 'animation'... to get better results."

Longer Queries: We also observed that our participants created longer queries as P6 added "record" to his keywords "graphs for plotting time" to narrow from 221 to 16 results and obtained more useful variants. These results led the participant to a program he used later to complete the task. It is also known that longer keyword queries return better search results [45].

Filter: FE participants used the filtering tools provided in File Exchange to narrow and enrich their variants. P8 used "Refine by Content Type" and "Refine by Product" and found the correct variant of the program in the first page of search results, which without filters, was placed on the second or third page of search results.

External Search to Narrow Internal Search: Participants also preferred using external search engines to help them narrow results in the internal search engines as P5 used Google to search "plotting a wave propagation on MATLAB file exchange". She later asserted preference towards external search engines, "*usually Google gives better results than directly searching in File Exchange*".

***Navigational Strategies*:** After obtaining search results, participants navigated the results using the following strategies:

Recency: Recency was used like a filter to narrow the results to more relevant variants for both AIG and FE participants. For example, P1 mentioned that she explored options like popular and recent apps (variants) because she expected that the browser history would remember the searches made by previous participants in the study. Another participant, P6, noted how an old app "*may not be so good*".

Ranking of Results: Rankings, in which variants appeared in the search results, were frequently utilized by both AIG and FE participants to narrow the search results of variants. For example, some participants expected the "*most relevant*" or "*best results*" (as P7 and P6 mentioned) to appear in the front/top of the page; this deterred participants from going to the next page of search results. As a result, most AIG and FE participants stayed on the first page of the search results to forage for variants.

Staying within a current resource: Three participants (P2, P3 and P6) were afraid of taking risks and stayed within the first variant they found or within patches as can be seen in Fig. 6. AIG participants lacked an understanding of the environment and the language. Therefore, when not enough information was available, participants stuck to the current resource instead of looking for additional resources outside of the environment.

Best variant selection: Participants tried to search for a variant that was the closest match for the task and improved upon it instead of finding and selecting relevant code snippets from less relevant variants. This is evident as P6 commented *"I checked an app two, three, four times. Just to see and I wanted to be sure that I was picking the best that would make my job easy".*

4.2.2. Evaluate destination variant

Participants evaluated a destination variant to determine whether it was within the context of the current task.

***Cues*:** While evaluating the variants, participants often used cues as signposts to develop stronger or weaker scents to select a variant or navigate away from variants. Our participants used the following cues while evaluating a relevant variant:

Visual Cues: Visual cues denote the images and snapshots associated with variants. Usually, these images present themselves along with a variant's details in a search result. The use of visual cues (images) was mentioned by six participants as a factor to explore the program further. These cues may depict a snapshot of the project, an icon of the program's function, which may be irrelevant to the project or nonexistent. For example, participant P2 found variant images relevant to their task and commented, "*this [application] looks [good] for me, the microphone.*" Three participants mentioned visual cues were their primary method of variant evaluation as P3 admitted he evaluated the variants by "*pretty much only the pictures.*" However, not all visual cues were appreciated equally; as P5 stated, "*the picture cannot really tell you the whole simulation of the code, so it's not that helpful.*"

Name-Inspired Cues: Name-Inspired cues represent the variant names/titles given by developers. All participants mentioned they used titles to guide them and five used it as their primary cue to evaluate a variant while the other three stated they used it as a secondary cue. This is similar to file-name inspired cues [28].

Description-Inspired Cues: Description-Inspired cues refers to the text describing the functionality of the program. Participants used descriptions of the variant or its description page to evaluate program functions. All FE participants used this cue, which gave a better understanding of the program's functionality. AIG participants used the

descriptions less, as P1 remarked AIG had "*short descriptions*", while FE had detailed descriptions.

Timestamp-Inspired Cues: Timestamp-Inspired cues refer to the date of creation or update of a project. In FE, these cues could be found below the images of the variant in gray. We found only one participant who mentioned the Timestamps. P6 expressed skepticism about the compatibility of a file by saying, "*if it's an old app, it may not be so good with MATLAB.*" He consequently searched "*to see if there's any update on the app.*"

Familiarity Cues: Familiarity cues are re-exposure to a variant through different query results. P6 explained he returned to an app because "*[the application] came three different times with three different keywords*". Sometimes these cues were misleading as P7 tried to revisit a project but did not realize it was the project he had already seen until he "*read the description*".

Social-Aspect-Inspired Cues: In online repositories, variants are associated with the social aspects of a project. Our participants mostly focused on the following aspects:

Authors: Participants used author/developer information to evaluate a variant. AIG and FE online repositories provided a list of projects created by the same author. P4 exploited this by evaluating more variants created by the same author, while P8 associated an image with its author and contemplated whether a project had the same author because of similar images.

Number of Functions: Participants evaluated variants as having too many or too little functions. For example, participant P1 abandoned a variant, commenting, "*it has functions that we don't want so I didn't go any further*".

Ratings: Variant ratings were judged superficially. P6 mentioned the rating system influenced him and explained that in FE, "*usually people do ratings according to [how closely the application matched the keywords searched]*". However, most participants did not pay attention to the ratings as P3 commented on not paying more attention to the "*eyeball*" (views) and the "*hearts*" (ratings). The ratings also confused participants as P6 lamented, "*the first option was like five stars, the second was zero stars . . . I wouldn't expect that to be on the list, actually.*" We can infer the participant expected the results to be sorted by ratings.

Hunting Strategies: When a participant's goal (predator's diet) changed while hunting for a variant [26], we observed the following strategies:

Prioritizing Cues: Participants focused on different cue types in a set order. To evaluate a variant, they used a primary cue then a secondary cue and so on. P4 said, "*first thing I looked at is the name,*" then said, "*apart from the name, I look at the pictures.*" The participants prioritized which cue to focus on in their between-variant search.

Sometimes participants reprioritized the cues while they searched. P8 said, "*pictures tend to be [attention] grabby, but a lot of them didn't have them so I had to go to the title*" then changed his order of the cues accordingly.

Creating/Using Scenarios: Participants empathized with the personas and used scenarios to decide if an application satisfied the task requirements. We found that AIG participants empathized with the Bob persona. P3 considered Bob's visual impairment and focused on visual elements to evaluate the variant as he commented, "*if I was picking an app for Bob who can't see well, this [app] has the biggest numbers".*

Since there were no scenarios for FE participants, they created personas according to different situations. P1 empathized with the developer, commenting he was "*[looking for] different words to search. . . [a developer] had used to post.*" Here we noted that our participants did not create any stories over time as found in [28].

***Navigational Strategies*:**

Depth-First Strategy: Depth-First strategy occurs when a participant understands the variants before acting on the results. All AIG participants stayed on one tab to understand and evaluate one variant at a time before going to the next variant. Only one FE participant showed this behavior.

Breadth-First Strategy: In Breath-First strategy, participants understood little information before acting and later sought more information, if needed. Before further understanding the variants, three FE participants opened multiple variants in new tabs to evaluate or download the patches. We found participants evaluated all the variants before choosing which patch to open or they evaluated each variant and opened/downloaded patches regardless of how many variants were opened.

Keeping Trails of Variants: After navigating/evaluating variants, only FE participants kept trails of desirable variants by leaving variants open. For example, P8 switched between different tabs of variants and commented "*I'll look it up here so I don't lose any of the programs I've found*".

4.2.3. Find destination patches

Inside the variant, participants found patches to modify the program. In within-variant foraging, the two repositories were different. AIG participants had three patch choices, as denoted in Fig. 1. In contrast, variants from FE may have contained several patches such as the actual code files (.m), installer files, and README, which were usually presented in file folders and were trivial for most users to navigate. In FE, three participants focused on .m files and only one participant focused on installer files.

4.2.4. Evaluate destination patches

Each patch found was evaluated based on the cues. Here is the list of the cues our participants followed:

***Cues*:** Table 4 shows all the cues used by our participants while evaluating a destination patch. Output-inspired cues were the most popular cue used to gather information. Code-inspired cues were studied by FE participants as they "*read line by line*" to understand the code according to P5. In the current context, internal documentation cues — called Doc.: Internal, namely tool tips and code comments, were used to understand the patch. Few participants used code-status-inspired cues which are elusive and hard to find [28].

Output-Inspired Cues: Output-inspired cues are features from the output of code generally seen in the emulator patch in the App Inventor environment and in the output window in the MATLAB environment. 88/131 of these cues were used by AIG participants, while 43/131 were used by FE participants. In AIG, participants used output-inspired cues from a patch to evaluate a selected variant. For example, P4 could not find any output cues in AIG so he backtracked to find a new destination variant. Participants in FE were particularly resourceful in the output patch. For example, P7 explored the file menu options in the output GUI and evaluated the tools that were available to determine whether MATLAB could perform his desired functionalities.

Code-Inspired Cues: Code-inspired cues are code segments of the program, which can be processed to determine if the patch should be a destination patch. For AIG 31/72 participants used code-inspired cues, while 39/72 FE participants used these cues. In the code editor, AIG participants read the labels on the blocks of code as clauses and sentences. P3 read a block as: *"When Clockon Timer"*. The FE participants primarily searched linearly through the code. P5 said, *"I...read line by line to know what it is exactly doing"*.

Documentation-Inspired Cues: We consider documentation as either internal or external to the patch; in the current context, we focus on internal documentation. (Note: External documentation is primarily used later in usage context during Stage 2.) We consider internal documentation to be either tooltips or code comments. AIG participants used 30/73 cues while FE used 43/73 internal documentation-inspired cues. AIG focused mainly on tooltips (28/30) and FE focused mainly on comments (34/43). This cue was mostly for understanding the patch.

Code Status-Inspired Cues: Cues which indicate whether the status of the program is correct or erroneous. Only two participants noticed these cues, 6/7 of these cues were used by one participant. Not having seen the error previously, P4 expressed *"'Show Warning'? That's something*

Table 4
Destination patch evaluation cues (Columns in gray denote FE participants).

	P1	P2	P3	P4	P5	P6	P7	P8	Total
Output	3	42	42	1	9	28	1	5	131 (53.3%)
Code	10	11	10	2	4	0	28	7	72 (29.3%)
Doc.: Internal	2	0	0	0	6	1	8	19	36 (14.6%)
Code status	1	6	0	0	0	0	0	0	7 (2.8%)

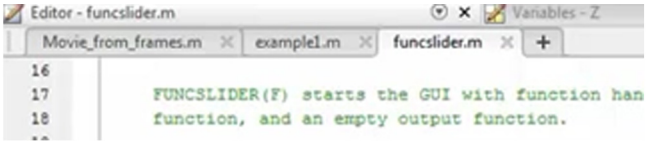

Fig. 5. Keeping trails of patches strategy.

new". We can infer these cues are, as Ragavan et al. [28] explained it, elusive — cues which are difficult to find.

Navigational Strategies:

Keeping Trails of Patches: Keeping trails of patches can be defined as when our participants kept a history of desirable patches. After evaluating patches, only FE participants kept trails of patches by keeping the patches or folders holding the patches open. Fig. 5 shows the behavior of P5 while using this strategy.

4.3. Stage 2: Finding and evaluating a usage context

In the usage context, a predator (EUP) forages between-variants to find and evaluate variants/patches to establish a source variant/patch to facilitate their current task.

4.3.1. Find and evaluate source variants

After finding and evaluating the destination patch (in Stage 1), AIG and FE participants exhibited different behaviors to find a source variant. All AIG participants except P4 stuck to the destination variant they chose, while FE participants actively searched and evaluated other source variants. FE participants used similar strategies and cues to find and evaluate the source variant as they did with the destination variant in Stage 1, except they enriched their results by using keywords from the task.

4.3.2. Find and evaluate source patches

FE participants used similar strategies to find and evaluate source patches as they did in Stage 1. Within the variants, FE participants utilized similar strategies to find the destination patch as they did in the current context. However, under the usage context, participants searched for different types of patches, mostly external documentations such as forums, Stack Overflow, MATLAB Answers, wikis, or tutorials etc. Thus, not only did the FE participants use the same cues as they did in the current context; they also utilized the *external documentation-inspired* cues due to the change in diet (goals) for an external documentation patch. The change was mainly driven by their desire to learn more about the task before utilizing or integrating the patch for use, which confirms the previous work on Minimalist Learning Theory (MLT) [46–48]. MLT states that programmers sometimes need to learn the code that they have come across while foraging to complete a task. For example, while attempting to finish the task, P6 came across code that he was not familiar with. Instead of immediately using the code, P6 learned about the code through *external documentation-inspired* cues to look for an example tutorial on the application as he commented, "I wanted to know how the app works… before I do something". Additionally, participants used external documentation to evaluate patches for desirable resources (e.g., code snippets) to utilize. Two participants tried to search for useful information in question-and-answer forums. P5, using Stack Overflow, said, "*I need an example for time varying 3D function so I can use the example [in the file code]*".

4.4. Stage 3: Integration

Integration occurs when the destination patch is modified by adding contents from other sources such as the EUP's knowledge base of another patch. The participants used the following strategies while integrating their code:

***Writing Code from Scratch*:** This process occurs when a programmer writes code within the programming environment. All AIG participants wrote code from scratch once they selected a destination patch, which lead to more Alternative Ideas (AIG 143 vs. FE 13). Meanwhile, FE participants wrote code from scratch after evaluating a variant and they did not stick to the destination patch and foraged for other variants. Thus, they had lower alternative ideas but higher search numbers (AIG 59 vs. FE 68). For example, P7 coded from scratch before finding a destination patch, while P5 coded from scratch after finding a destination patch related to the task.

***Reusing Code*:** When participants integrated or imported code from other source patches to a destination patch, they reused code. There are two types of reuse as defined by Holmes and Walker: analyze-then-act and cut-and-stanch the bleeding [2].

Analyze-then-act analysis: This type of reuse involves locating the source of reuse for the most useful code(s) then integrating code in a destination patch (copy and paste, no modification). FE participants analyze-then-acted mostly from external documentation. We observed two reasons when reusing code—to add contents and modify to the participant's need or to understand how the code works [41].

Cut-and-stanch the bleeding: In this process, participants take the relevant code and integrate it into the destination patch. AIG participants only used cut-and-stanch the bleeding strategies because they stuck to the one patch they found, and they integrated within the project by copying the existing code and modifying it for their use (copy and paste, modification).

5. Discussion

In this section, we discuss the foraging behavior of all participants during the study and compare the cues that were specifically used by the novice and experienced EUPs.

5.1. Timeline of participants' behaviors

Fig. 6 summarizes the foraging behaviors of all participants denoting cues used during each stage. The outermost colors describes the three major stages in foraging as discussed in the Reuse Model: between foraging, within foraging, and integration. Additionally, to better identify the participants' foraging timeline and to organize the user's foraging behaviors during the two stages—Current Context, Usage Context—we designate the colored middle bar with eight different

minor stages. This further breaks down the stages into actions, foraging for either a variant or a patch by their current or usage context (refer Fig. 3). In the current context, participants forage for a destination variant. The four minor stages of this process are foraging destination variants (DVF), evaluating variants (DVE), foraging patches (DPF), and evaluating patches (DPE). Similarly, in the usage context, participants forage for a source variant and a patch. The four minor stages are foraging source variants (SVF), evaluating variants (SVE), foraging patches (SPF), and evaluating patches (SPE). We observed the following patterns:

Between- vs. Within-Variants: All participants went from foraging between-variants to foraging within-variants as shown in Fig. 6. FE participants foraged between-variants more frequently than AIG participants. We conjecture this was because FE participants did not have enough cues associated with a variant to evaluate it and hence used more cues while evaluating patches. Meanwhile, AIG participants had more cues associated with the variant and needed less cues in the patch to evaluate it.

***Source vs. Destination*:** AIG participants used less (four) different cue types in the source as compared to FE participants (seven). Seven of the participants spent less time foraging in the source than in the destination (refer Fig. 6). Only P8 spent more (>50%) time in the source than in the destination.

Six participants went straight to integration after destination (refer Fig. 6). Only P5 went to integration after the source because she had code errors, and rather than returning to the destination, she went to the source. Later when asked about this behavior, she commented "*google gives better results than directly searching in File Exchange*".

***Order of Stages*:** Participants went through stages in different orders. Fig. 6 shows that P5 foraged from the integration stage and went back to the between-variants stage. This behavior was different from other participants as, once they were in the integration stage they would not leave it. Only P5 and P7 went from foraging between-variants to integration, and compared to the other six participants, P7 foraged between-variants the most. As observed in Fig. 6, participants varied their stage paths from one another, and there is not one specific way of moving between stages.

Staying within a Stage: FE and AIG participants remained within a stage for various amounts of time. We conjecture this behavior is driven by the predator's wish to optimize the value per the cost when evaluating variant/patches' scents (as discussed in Section 2.2).

Three AIG participants (P1, P2 and P3) stayed in the within forging and integration stages to understand the patch and complete the task. As none of the AIG participants had experience in the environment, they were more risk-averse. This resulted in lower expectations for value gained compared to the higher cost in transitioning between stages. For example, P4 who foraged between-variants had to search the apps using appropriate keywords, select the appropriate app, open that app, and then evaluate it. When foraging variants/patches the cost incurred was more for foraging between-variant/patch thus participants stayed in within-variant/patch forging or the integration stage as seen in Fig. 6.

Meanwhile, most FE participants (P5, P7, and P8) utilized all three stages—the between, within, and integration—to forage for their prey. This behavior was representative of their better knowledge of the environment, which drove higher value gained with lower cost during transitioning.

5.2. Comparing cues used by novice and experienced EUPs

In both AIG (novice) and FE (experienced) groups, the type of cues used varied as seen in Table 5. FE participants used more description-inspired cues than AIG participants because the AIG Repository had more images than the FE Repository. AIG participants found images next to the title of the variant helpful, while FE participants found that images (visual cues) were not representative of the functionality of the variant. For example, P3 commented, "*... pictures. ... that's what it [app] would look like on the screen... I didn't pay any attention to username, or emails down there...* ", while P7 commented, "*...the plot, the picture was way more than what I was trying to do...so I just ignored those.*" Hence, AIG participants assumed that the images produced the output for an app, but FE participants ignored the images and relied more on the description and title of an app.

Cues associated with finding variants: We observed that while finding variants, both FE and AIG participants used description-inspired, name-inspired, and visual cues. We also observed different cues being used by FE and AIG participants. FE participants used document-inspired cues and code-inspired cues while AIG used output-inspired and author cues.

Cues associated with evaluating variants: While evaluating variants, both FE and AIG participants used description-inspired cues. In addition, FE participants used document-inspired cues while AIG participants used name-inspired cues.

Cues associated with finding patches: Both FE and AIG participants used description-inspired cues while finding patches. In addition to description-inspired cues, AIG participants used name-inspired and author cues while FE participants did not.

Cues associated with evaluating patches: We observed that while evaluating patches, all participants used output-inspired cues. FE participants used visual and description-inspired cues in addition to output-inspired cues.

Cues associated with transitions: Different cues were associated with different stage transitions (refer Table 6). For example, due to little description being available in the AIG repository, the AIG participants relied more on name-inspired cues, while for the FE participants, the description-inspired cue caused the change from the DVF stage to the DVE stage. For transitioning from the DVF stage to the DVE stage, AIG participants used the name-inspired cue six times while FE participants used the description-inspired cue seventeen times. We conjecture that this may be because these were clear cues (easy to understand) [26] and led to stronger scents. Further, these cues may have led them to patches with more information features.

Cues associated with destination: We observed in destination both FE and AIG participants used description-inspired cues (refer Table 6). AIG participants used more name-inspired cues than FE participants, and FE participants used more output-inspired cues than AIG participants. AIG participants also used author cues while FE participants did not.

Cues associated with the source: While in the source, both FE and AIG used visual and name-inspired cues (refer Table 6). AIG participants did not use any other cues besides these two cues.

6. Implications

In this section, we discuss implications for IFT and tools for EUPs based on our results.

6.1. Implications for theory

We already know that IFT can model dissimilar but connected patches. In current programming scenarios, while reusing code, an EUP may find code snippets on different websites (Stack Overflow, documentation, blog post etc.) or in a code repository from programs written by the same or/different authors. This opens research opportunities to extend the IFT model to include similar variants/patches or mechanisms to organize the similar variants/patches into a hierarchy.

Cues and Strategies of EUPs in Online Repositories: Since we analyzed EUPs' behavior in online repositories, we found new sets of cues (noted previously in results) that had not been discussed in previous IFT literature. In Ragavan et al.'s study variants were associated with Timestamps (cues), whereas in our study, the online repositories had more cues associated with the variants. As a result, our participants used cost–benefit analysis to evaluate both variants and patches. Among the new cues, social aspect and familiarity-based cues were

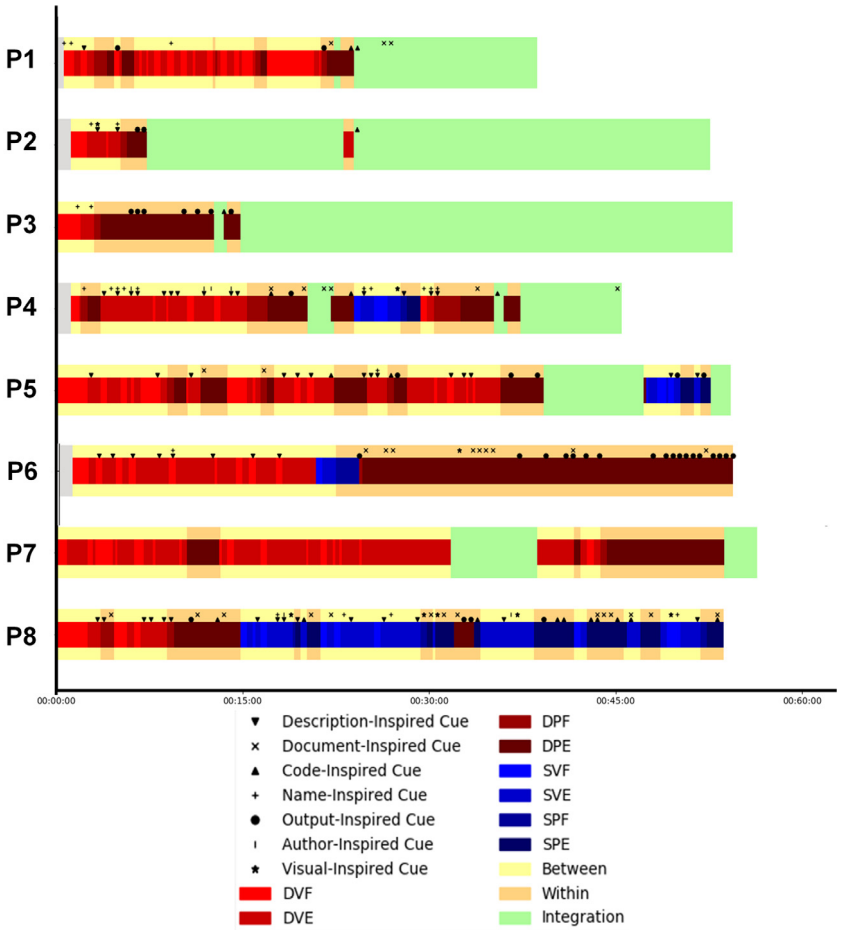

Fig. 6. Participants' stage patterns and associated cues.

Table 5
Cues associated with stages.

Cues:		Stage	Description		Doc		Code		Name		Output		Visual		Author	
AIG	FE	DVF							1							
		DVE	10	10					5	1			1		4	
		DPF														
		DPE			3	8	4	1			9	21		1		
		SVF								1						
		SVE		6					2	2			1	2		2
		SPF														
		SPE				6		5				2				
		INT			5		1									

Table 6
Cues Associated with Stage Transitions.

Cues:	Description		Document		Code		Name		Output		Visual		Author	
Destination	AIG	FE	AIG	FE	AIG	FE	AIG	FE	AIG	FE	AIG	FE	AIG	FE
DVF<>DVE	4	17					6	1	1					
DPF<>DPE			1	2	1	1			1	1				
DVF/DVE<>DPF/DPE	2	3				1	3		1				1	
Cues:	Description		Document		Code		Name		Output		Visual		Author	
Source	AIG	FE	AIG	FE	AIG	FE	AIG	FE	AIG	FE	AIG	FE	AIG	FE
SVF<>SVE	1	4		1			1	1				1	1	
SPF<>SPE		1		1										
SVF/SVE<>SPF/SPE	1	2		2		3				1		3		
SPE>DVF							1							
SPE<>DPE				1		1								

more popular with participants to make variant foraging decisions in online repositories.

The cues also influenced the foraging decisions of the EUPs. For example, we found that description cues led the transition between

finding and evaluating variants the most. However, for patch finding and evaluating, there were no description cues used but instead document, code, and output cues. Thus, to encourage movement between the different stages, file repositories should emphasize the cues that cause stage transitions and limit the other cues that do not. This will allow users to reach their end goal faster and reduce the time they are in a stage. By prioritizing cues and emphasizing the cues that do lead to transitions, users will not be lost by the many cues presented to them in a stage. Hence, it motivates the need to operationalize different cues when variants/patches are involved in the IFT model.

Similarly, we found differences in the strategies followed during different stages. External documentation is used to inspire learning as an external documentation-inspired cue but can also be used as a patch for fixing—also known as integrating in our model. This behavior is similar to that of professional programmers found by Piorkowski et al. that there is a difference in learning and fixing code [39]; learning is similar to the finding and evaluating behavior of participants of the source variant/patch and fixing is similar to working on code from a destination variant/patch. This suggests that the IFT model should dynamically operationalize cues, variants, and patches based on EUPs' needs at a particular time.

Comparison and Bridging Gaps in Knowledge: Comparison and bridging gaps in knowledge are categorized as overall strategies in the reuse model because they were used and can be used anywhere between- and within-variants as well as in finding and evaluating variants and patches. In contrast, Ragavan et al.'s reuse model only exemplified strategies for foraging between- and within-variants.

Comparison was intriguing because only experienced EUPs, FE participants, utilized the strategy to compare not only between- and within-variants (variants to variants and patches to patches) but also across variants and patches (variants to patches). In other words, comparison could happen anytime in the Reuse models and should be introduced into the IFT model, reaffirming the findings of Ragavan et al. [28].

Bridging gaps in knowledge was also interesting because both groups of participants–novice and experienced–utilized this to change their goals and strategies. Participants also switched between navigation and enrichment strategies to further bridge gaps in knowledge. The goals or strategies of users can change anytime in the IFT model due to the gaps of their knowledge.

Navigator vs. Explorer Predator: Our results show that the two groups of participants (predators) exhibited different types of characteristics, namely: navigators and explorers. AIG participants were navigators who were more risk averse and thus mostly navigated within their given environment (AIG repository and IDE), utilized few strategies, and stuck to their source variant/patch and did not look for variants or patches in the usage context. Meanwhile, FE participants were explorers who took risks and used both internal and external search engines, utilized various strategies, and searched for more variants and patches in usage context. Hence, IFT should model the navigator and explorer behaviors of predators to fit their specific needs.

6.2. Implications for tools

Removing/Adding Search Trails: While evaluating or understanding variants/patches, we found that the participants kept a trail mentally or physically when comparing across variants/patches. Online repositories do not support this comparison behavior. Further, we also found that participants wanted either some variant to be included or removed from the list of variants returned by searches in online repositories. Thus, they wanted to keep trails or remove the uninteresting trails from their history. These findings suggest that there is a need for tools to support such behavior which will help in reducing the time for finding similar variants and cognitive load of EUPs.

Integrating Enrichment Strategies: Our participants used different enrichment strategies while searching for the variants in online repositories. Currently, most of the end-user internal-code search engines are based on keyword search. The keyword-based search engines need EUPs to recall the information related to the variant they are looking for, hence adding more cognitive load. There is a need for better search engines, which may facilitate EUPs to just recognize the variants. Some of the search engines like specification-based searches [49], CODEBROKER [50], or behavior-based clustering for visual programs [51] can be helpful to an extent.

Current keyword-based search engines can utilize query formulation strategies and integrate them to formulate better queries to get better search results. As we have seen, novice EUPs did not use most of the enrichment strategies. This motivates the need for search engines to facilitate query formulations, like facilitating the formation of longer queries [45], giving recommendations for query formulations [52], and allowing filters to be used efficiently.

Automatic Cues Extractions: Our results suggest having different cue types, visual and textual (name-inspired and description-inspired), will be helpful for both types of EUP groups (novice and experienced). Hence, online repositories as well as IDEs should facilitate the automatic extraction of the outputs to give better visual cues and of the features to give better textual descriptions of a program. The results also suggest that having more cues of one type will cause users to focus on that one cue type, think highly of those cues, and neglect other cues that do not show up as often. Having a balanced number of cues of different types will reduce the user's over thinking in excess cues and negligence in few cues. These cues should be associated with the variants to facilitate evaluation of variants/patches.

Allowing Explorations: As previously mentioned, we found that the AIG participants were navigators who stayed in the IDE once they found the relevant patch. They never looked for the relevant variants they needed to integrate. This type of behavior can be facilitated by having a recommendation system which can recommend variants based on previous users' actions (i.e. "people who viewed this also viewed this…"). Additionally, having the search engine integrated within an IDE can allow for easy access to relevant variants (i.e. within the same window, the IDE and search engine can be open side-by-side).

Finding Code Using Code Properties: The current search mechanisms for internal code repositories and external search engines both tailor to text-based search. Code is inherently different from the text description—code has various properties like the number of parameters in a function, data types, behavior, readability, and correctness that current search engines do not fully enable foraging for. As noted, our participants used text descriptions to describe the semantic of the code while searching for a variant (apps/projects), and depended on existing descriptions for variants, which resulted in a higher margin of error. This also increases cognitive load for EUPs as they align their strategy to the current text-based search engine. Further, this challenge is more apparent in visual programming languages which are not easily accessible using the current text-based search engine. This necessitates the need for researching more sophisticated mechanisms that utilize the plethora of code properties (i.e. using the number of parameters in a function, data types, etc. in addition to the given text descriptions) within search engines.

7. Threats to validity

All empirical studies have threats to validity that should be considered when interpreting results. In this section, we present the threats to validity of our study.

7.1. External validity

Our study participants are representative of a small sample of the EUP population, leading to an external threat to validity. Furthermore, within this small sample, the gender was not balanced as only two participants were female. Hence, the behaviors of our study participants

may not be representative of how all EUPs may forage in online repositories or integrate between patches. However, this was a formative study and we used two different groups of participants to get a broad understanding of EUPs and their behaviors. For our study, we used volunteers who were motivated to complete the tasks, as shown by their willingness to participate. Additionally, for convenience, we selected from a student sample which can be equivalent to professionals "when their knowledge, skills and experiences fit within the tool's intended user population" [53]. Overall, our sample of EUPs are in accordance with (1) Nielsen's scale for usability studies [54], (2) Johnson's minimum required population for controlled experiments [55], and (3) studies on end-user software evaluations [e.g. 56,57].

Another threat can arise from the choice of the two groups used in our study, the AIG group consisting of novice EUPs and the FE group of experienced EUPs. Since we evaluated only these two groups, the results on the foraging behavior of EUPs may have been influenced by the domain knowledge, complexity of the foraging environment, and experience and skill level of the participants. Although, this was an intentional study design choice to allow us to analyze programmer experience as a factor while foraging in online repositories.

Furthermore, the tasks given to participants were only a small sample of possible tasks and may not be representative of real-world scenarios. Although, the tasks were formed based on the popularity of variants in the AIG and FE repositories and we attempted to create scenarios that seemed realistic.

7.2. Internal validity

In our study, we designed two tasks to be similar across the two selected environments, AIG and FE. In both cases, Task 1 consisted of searching for a program variant given a set of specifications and Task 2 consisted of finding another program to integrate into the one found in Task 1. We intended for the resulting found variants to be on the second page of the participants' searches despite the environment. The two tasks were not counterbalanced which could have led to learning effects regarding the tasks as well as foraging mechanisms used by the participants. Although, this design was intentional as the second task was meant to depend on the first task in order to mimic a real-life reuse scenario where searching/finding, evaluating, and selecting is performed before integrating reusable code into another code base.

7.3. Construct validity

Despite attempts to make the two tasks similar across different environments there is a construct threat to validity as the complexity of the tasks may not have been equivalent for each (AIG and FE) group of participants.

8. Related works

In this section, we discuss the related works on code reuse in online repositories and IFT as utilized by the software engineering community.

8.1. Code reuse in online repositories

Code reuse in online repositories can be a challenging task as the information seeking behavior of EUPs requires a higher cognitive effort and may not be fully supported by the tools provided within online repositories. EUPs' code reuse can be facilitated through the implementation of tools that support the finding and reusing of appropriate variants of a program. Variation-supporting tools have been utilized for comparing and creating code alternatives [58–61]. Some of these tools consider a single project and facilitate the variant creation process. Other tools facilitate the variant selection process by comparing variants graphically. Hartmann et al. introduced Juxtapose as a tool for parallel code editing to quickly create code alternatives and simultaneously compare their output [62]. Kuttal et al. implemented AppInventorHelper, a variation management support tool, to help EUPs visualize the relationships among variants. This tool had provided insights into EUPs' variation selection behavior and proved to be useful in navigating through variants and determining which to reuse [63]. In addition to these tools, great strides have been made in file management in regards to variation: Karlson et al. introduced the "version-set", a representation of variants from a single source, to help users better manage their personal information (i.e., their personal repository) [64]. Kuttal et al. created tools to allow end-user programmers to keep a collection of "past and present variants" originating from a single artifact [63,65–69]. Furthermore, to assist in the analysis of foraging among variants, Ragavan et al. introduced PFIS-V, a computational model that models people's forging behavior within an environment containing variants [29]. Such a model can be utilized to provide implications on how to improve existing tools dealing with variants and what other tools may be necessary.

8.2. IFT in software engineering

Software engineering has benefitted from IFT. The domains include debugging, code reuse, code maintenance, and navigation behavior modeling and prediction [19–23]. Lawrance et al. presented a model, the PFIS2, which suggests navigation tools can predict appropriate places to guide programmers towards fixing bugs in their code [23]. This model was further progressed by Piorowski et al. to understand programmers' goals and strategies as they forage to debug their programs [24]. Later, Piorkowski et al. developed PFIS3 and introduced a PFIS-based tool for debugging [25]. Kuttal et al. studied debugging behavior by using IFT to examine web-active EUPs' interactions and provided categories for the cues and strategies utilized by EUPs when finding and fixing bugs [26,27].

Previous literature related to IFT focused on foraging behavior in a single artifact (variant). Ragavan et al. extended the model to include the foraging behavior of novice programmers when given temporally similar variants [28]. Inspired by their work, we utilized their model to understand how EUPs forage for similar variants not only temporally (variants of same project developed over time) but also spatially (variants of similar projects developed by different authors).

9. Conclusion

This is the first study to use an IFT perspective to examine the reuse behavior of EUPs in online repositories. Overall, our participants changed their goals and strategies to bridge gaps in knowledge by reassessing their knowledge base. Further, they compared source variants/patches all throughout the reuse phenomena. Our results revealed the following insights:

RQ1 - Types of information used In variants, our results showed new cues such as: visual-inspired, description-inspired, social-aspect-inspired, and familiarity-inspired. Meanwhile, in patches, we found cues such as: documentation-inspired, output-inspired, code-inspired, and code-status-inspired. Additionally, experienced programming participants preferred textual cues while other participants preferred visual cues while evaluating variants to consume.

- ***RQ1a - Between-variant foraging*:** Participants used various foraging strategies to reduce the cost of finding and evaluating a variant. They used enrichment strategies such as: domain-specific keywords, filtering results, using external search engines to narrow results in internal search engines, recurring semantically-similar keywords, and longer queries. After enriching variants, all participants used navigational strategies such as: recent variants and ranking of variants. They also used hunting strategies such as: prioritizing cues and empathizing with personas or developers to find the variant. Participants with more programming experience utilized more diverse strategies to find and evaluate a variant.

- ***RQ1b - Within-variant foraging*:** We found that experienced programming participants used keeping trails strategy to keep a history of desirable patches. Further, more experienced programming participants used different variants, patches, external documents to evaluate a patch. Meanwhile, less experienced participants limited themselves to a single variant and explored the internal documentation as all the patches in the variant had to be utilized.

***RQ2 - Integrating code*:** Participants used strategies such as: writing code from scratch, analyze-then-act, and cut-and-stanch. Experienced programming participants used writing code from scratch and analyze-then-act strategies to better understand the destination and source patches to facilitate integration. Meanwhile, less experienced programming participants did not utilize source variant/patch and therefore used writing code from scratch and cut-and-stanch strategies to integrate their programming task which resulted in many alternative ideas.

Finally, we discuss implications for IFT theory and reveal new opportunities to design better tools for EUPs who reuse code variants in online repositories.

CRediT authorship contribution statement

Sandeep Kaur Kuttal: Supervised throughout the life of the project, Designed and conducted the user studies with eight participants, Analyzed the transcripts, Reviewed and wrote the paper. **Se Yeon Kim:** Analyzed the transcripts in-depth, Wrote the initial draft of the paper. **Carlos Martos:** Created the transcript and collected the code set, Wrote the initial draft of the paper. **Alexandra Bejarano:** Helped with writing and reviewing the final draft.

Declaration of competing interest

Following are the researchers I have collaborated with on research papers or research grants.

Allen, Joseph: University of Nebraska at Omaha; Rothermel, Gregg: University of Nebraska- Lincoln; Sarma, Anita: University of Nebraska-Lincoln; Burnett, Margaret, M.: Oregon State University; Chen, Xiaofan: Newenergy Enterprise Group Limited, NZ; Dabbish, Laura: Carnegie Mellon University; Fleming, Scott: University of Memphis; Gamble, Rose: University of Tulsa; Hale, Mattew: University of Nebraska at Omaha; Kwan, Irwin: Mathworks, USA; Bellamy, Rachael: IBM Research, USA; Peters, Anica: Polytechnic of Namibia, Namibia, Africa; Piorkowski, David: IBM Research, USA; Lee, Michael, J: New Jersey Institute of Technology; Myers, A. Brad: Carnegie Mellon University; Ko, Amy: University of Washington; Ahmed, Iftekar: University of California, Irvine; Eziquel Scott: University of Tartu, Estonia; Sharma, Rajesh: University of Tartu, Estonia; Yunfeng Zhang: IBM Research, USA; Sruti Ragavan: Microsoft Research, England

Acknowledgments

We would like to thank Jiayi Lu and Cao Huyng for their contributions to the study analysis and our study participants for their valuable time.

References

[1] J. Brandt, P.J. Guo, J. Lewenstein, M. Dontcheva, S.R. Klemmer, Opportunistic programming: Writing code to prototype, ideate, and discover, IEEE Softw. 26 (5) (2009) 18–24.
[2] R. Holmes, R.J. Walker, Systematizing pragmatic software reuse, ACM Trans. Softw. Eng. Methodol. 21 (4) (2013) 44, Article 20.
[3] M. Burnett, B. Myers, Future of end-user software engineering: beyond the silos, in: International Conference on Software Engineering (ICSE) Companion Proceedings, 2014, pp. 201–211.
[4] MathWorks. File Exchange - MatLab Central. Retrieved June 26, 2020 from http://www.mathworks.com/matlabcentral/fileexchange/.
[5] MathWorks. MATLAB - MathWorks - MATLAB & Simulink. Retrieved June 26, 2020 from http://www.mathworks.com/products/matlab/?requestedDomain=www.mathworks.com.
[6] MIT App Inventor. Join the App Inventor Community Gallery. Retrieved June 26, 2020 from http://appinventor.mit.edu/explore/blogs/shay/2013/04/join-app-inventor-community-gallery.html.
[7] MIT App Inventor. MIT App Inventor | Explore MIT App Inventor. Retrieved June 26, 2020 from https://appinventor.mit.edu/.
[8] Scratch. Scratch - Explore. Retrieved June 26, 2020 from https://scratch.mit.edu/explore/.
[9] Scratch. Scratch - Imagine, Program, Share. Retrieved June 26, 2020 from https://scratch.mit.edu/projects/editor/?tip_bar=home.
[10] K.T. Stolee, S. Elbaum, A. Sarma, Discovering how end-user programmers and their communities use public repositories: A study on Yahoo! Pipes, Inf. Softw. Technol. 55 (2013) 1289–1303.
[11] S.K. Kuttal, A. Sarma, G. Rothermel, Debugging support for end-user mashup programming, in: Computer-Human Interaction (CHI), 2013, pp. 1609–1618.
[12] C. Bogart, M. Burnett, A. Cypher, C. Scaffidi, End-user programming in the wild: A field study of coscripter scripts, in: IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2008, pp. 39–46.
[13] W.E. Mackay, Patterns of sharing customizable software, in: ACM Conference on Computer-Supported Cooperative Work, 1990, pp. 209–221.
[14] K. Martzoukou, A review of web information seeking research: considerations of method and foci of interest, Inf. Res. 10 (2) (2005).
[15] W. Fu, P. Pirolli, SNIF-ACT: a cognitive model of user navigation on the world wide web, in: Human-Computer Interaction, Vol. 22, (4) 2007, pp. 355–412.
[16] P. Pirolli, W. Fu, E. Chi, A. Farahat, Information scent and web navigation: Theory, models and automated usability evaluation, in: Proceedings of HCI International, 2005.
[17] P. Pirolli, W. Fu, SNIF-ACT: a model of information foraging on the world wide web, in: User Modeling 2003, Springer Berlin Heidelberg, 2003, pp. 45–54.
[18] P. Pirolli, Computational models of information scent-following in a very large browsable text collection, in: Computer-Human Interaction (CHI), 1997, pp. 3–10.
[19] N. Niu, A. Mahmoud, G. Bradshaw, Information foraging as a foundation for code navigation (NIER track), in: International Conference on Software Engineering (ICSE), 2011, pp. 816–819.
[20] J. Lawrance, M. Burnett, R. Bellamy, C. Bogart, C. Swart, Reactive information foraging for evolving goals, in: Computer-Human Interaction (CHI), 2010, pp. 25–34.
[21] S.D. Fleming, et al., An information foraging theory perspective on tools for debugging, refactoring, and reuse tasks, ACM Trans. Softw. Eng. Methodol. 22 (2(14)) (2013).
[22] J. Lawrance, C. Bogart, M. Burnett, R. Bellamy, K. Rector, How people debug, revisited: an information foraging theory perspective, in: IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2009, pp. 117–124.
[23] J. Lawrance, R. Bellamy, M. Burnett, K. Rector, Using information scent to model the dynamic foraging behavior of programmers in maintenance tasks, in: Computer-Human Interaction (CHI), 2008, pp. 1323–1332.
[24] D. Piorkowski, S.D. Fleming, C. Scaffidi, L. John, C. Bogart, B.E. John, M. Burnett, R. Bellamy, Modeling programmer navigation: a head-to-head empirical evaluation of predictive models, in: IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2011, pp. 18–22.
[25] D. Piorkowski, S.D. Fleming, C. Scaffidi, C. Bogart, M.M. Burnett, B.E. John, R. Bellamy, C. Swart, Reactive information foraging: an empirical investigation of theory-based recommender systems for programmers, in: Computer-Human Interaction (CHI), 2012, pp. 1471–1480.
[26] S.K. Kuttal, A. Sarma, G. Rothermel, Predator behavior in the wild web world of bugs: an information foraging theory perspective, in: IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2013, pp. 59–66.
[27] S.K. Kuttal, M.M. Burnett, A. Sarma, G. Rothermel, I. Koeppe, B. Shepherd, How end-user programmers debug visual web-based programs: An information foraging theory perspective, J. Compu. Lang. (2019).
[28] S.S. Ragavan, S.K. Kuttal, C. Hill, A. Sarma, D. Piorkowski, M. Burnett, Foraging among an overabundance of similar variants, in: Computer-Human Interaction (CHI), 2016, pp. 3509–3521.
[29] S.S. Ragavan, B. Pandya, D. Piorkowski, C. Hill, S.K. Kuttal, A. Sarma, M. Burnett, PFIS-V: Modeling foraging behavior in the presence of variants, in: Computer-Human Interaction (CHI), 2017.
[30] C. Martos, S.Y. Kim, S.K. Kuttal, Reuse of variants in online repositories: foraging for the fittest, in: IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2016, pp. 124–128.
[31] P. Pirolli, S. Card, Information foraging in information access environments, in: Computer-Human Interaction (CHI), 1995.
[32] P. Pirolli, Information Foraging Theory: Adaptive Interaction with Information, Oxford University Press, 2009.
[33] P. Pirolli, Rational analyses of information foraging on the web, Cogn. Sci. 29 (3) (2005) 343–373.

[34] MathWorks. MATLAB vs. Python: Top Reasons to Choose MATLAB. MATLAB vs. Python: Top Reasons to Choose MATLAB - MATLAB & Simulink. Retrieved June 26, 2020 from https://www.mathworks.com/products/matlab/matlab-vs-python.html.
[35] C.H. Lewis, Using the 'Thinking Aloud' method in cognitive interface design, 1982, RC 9265, IBM.
[36] TechSmith, Morae. Retrieved June 26, 2020 from https://www.techsmith.com/morae.html.
[37] F. Shull, J. Singer, D.I. Sjøberg, Guide to Advanced Empirical Software Engineering, Vol. 93, Springer, 2008.
[38] P. Borlund, Experimental components for the evaluation of inteactive information retrival systems, J. Documentations 56 (1) (2000) 71–90.
[39] D. Piorkowski, S.D. Fleming, C. Scaffidi, M. Burnett, I. Kwan, A.Z. Henley, J. Macbeth, C. Hill, A. Horvath, To fix or to learn? How production bias affects developers' information foraging during debugging, in: International Conference on Software Maintenance and Evolution (ICSME), 2015.
[40] P. Jaccard, Étude comparative de la distribution florale dans une portion des Alpes et des Jura, Bulletin del la Société Vaudoise des Sciences Naturelles 37 (1901) 547–579.
[41] M.B. Rosson, J.M. Carroll, The reuse of uses in smalltalk programming, ACM Trans. Softw. Eng. Methodol. 3 (3) (1996) 219–253.
[42] B. Dervin, B.D.L. Foreman-Wernet, et al., Sense-Making Methodology Reader: Selected Writings of Brenda Dervin, Hampton Press, 2003.
[43] A. Abraham, Information Seeking from Web-Based Resources: Sensemaking Strategies and Implications for Interaction Design (Ph.D. dissertation), The Open University, 2013.
[44] J. Teevan, C. Alvarado, M.S. Ackerman, D.R. Karger, The perfect search engine is not enough: A study of orienteering behavior in directed search, in: Computer-Human Interaction (CHI), 2004, pp. 415–422.
[45] N.J. Belkin, et al., Query length in interactive information retrieval, in: Special Interest Group in Information Retrieval (SIGIR), 2003, pp. 205–212.
[46] J.M. Carroll, The Nurnberg Funnel, MIT Press, Cambridge, MA, 1990.
[47] J.M. Carroll, Minimalism Beyond the Nurnberg Funnel, MIT Press, Cambridge, MA, 1998.
[48] H. van der Meij, J.M. Carroll, Principles and heuristics for designing minimalist instruction, Tech. Commun. 42 (2) (1995) 243–261.
[49] K.T. Stolee, S. Elbaum, Toward semantic search via SMT solver, in: FSE, 2012, Art. 25.
[50] Y. Ye, G. Fischer, Supporting reuse by delivering task-relevant and personalized information, in: International Conference on Software Engineering (ICSE), 2002, pp. 513–523.
[51] S. Surisetty, C. Law, C. Scaffidi, Behavior-based clustering of visual code, in: IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2015, pp. 261–269.
[52] L. Martie, T.D. LaToza, A. van der Hoek, Codeexchange: Supporting reformulation of internet-scale code queries in context (t), Autom. Softw. Eng. (2015) 24–35.
[53] A.J. Ko, T.D. LaToza, M.M. Burnett, A practical guide to controlled experiments of software engineering tools with human participants, Empir. Softw. Eng. (2015) 110–141.
[54] J. Nielsen, How many test users in a usability study?, 2012, Nielsen Norman Group 4, 2012. Retrieved September 26, 2020 from https://www.nngroup.com/articles/how-many-test-users/.
[55] P. Johnson, Human Computer Interaction: Psychology, Task Analysis, and Software Engineering, McGraw-Hill, 1992.
[56] S. Arslan, G. Kardas, DSML4DT: A domain-specific modeling language for device tree software, Comput. Ind. 115 (2020).
[57] T. Miranda, M. Challenger, Baris Tekin Tezel, Omer Faruk Alaca, Ankica Barišić, Vasco Amaral, Miguel Goulão, Geylani Kardas, Improving the usability of a MAS DSML, in: International Workshop on Engineering Multi-Agent Systems, Vol. 11375, 2019, pp. 55–75.
[58] B. Hartmann, S. Follmer, A. Ricciardi, T. Cardenas, S.R. Klemmer, d.note: revising user interfaces through change tracking, annotations, and alternatives, in: Computer-Human Interaction (CHI), 2010, pp. 493–502.
[59] R. Kumar, J.O. Talton, S. Ahmad, S.R. Klemmer, Bricolage: example-based retargeting for web design, in: Computer-Human Interaction (CHI), 2011, pp. 2197–2206.
[60] M. Terry, E.D. Mynatt, Side views: persistent, on-demand previews for open-ended tasks, User Interface Softw. Technol. (2002) 71–80.
[61] M. Terry, E.D. Mynatt, K. Nakakoji, Y. Yamamoto, Variation in element and action: supporting simultaneous development of alternative solutions, in: Computer-Human Interaction (CHI), 2004, pp. 711–718.
[62] B. Hartmann, L. Yu, A. Allison, Y. Yang, S.R. Klemmer, Design as exploration: Creating interface alternatives through parallel authoring and runtime tuning, in: Proceedings of the ACM Symposium on User Interface Software and Technology, 2008, pp. 91–100.
[63] S.K. Kuttal, A. Sarma, G. Rothermel, Z. Wang, What happened to my application? Helping end users comprehend evolution through variation management, Inf. Softw. Technol. 103 (2018) 55–74.
[64] A.K. Karlson, G. Smith, B. Lee, Which version is this?: improving the desktop experience within a copy-aware computing ecosystem, in: Computer-Human Interaction (CHI), 2011, pp. 2669–2678.
[65] S.K. Kuttal, Leveraging Variation Management to Enhance End Users' Programming Experience (Ph.D. dissertation), ETD Collection for University of Nebraska – Lincoln, CS, UNL, Lincoln, NE, 2014.
[66] S.K. Kuttal, A. Sarma, G. Rothermel, On the benefits of providing versioning support for end users: an empirical study, ACM Trans. Softw. Eng. Methodol. 21 (2(9)) (2014).
[67] S.K. Kuttal, Variation support for end users, in: IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2013, pp. 183–184.
[68] S.K. Kuttal, A. Sarma, G. Rothermel, History repeats itself more easily when you log it: versioning for mashups, in: IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2011, pp. 69–72.
[69] S.K. Kuttal, A. Sarma, A. Swearngin, G. Rothermel, Versioning for mashups - an exploratory study, in: Proceedings of the International Symposium on End-User Development – IS-EUD, 2011, 25–41.