Reuse of Variants in Online Repositories: Foraging for the Fittest

Carlos Martos, Se Yeon Kim, Sandeep Kaur Kuttal
Tandy School of Computer Science
University of Tulsa
Tulsa, OK USA
{carlos-martos, seyeon-kim, sandeep-kuttal}@utulsa.edu

***Abstract*—Programming is a creative task and is generally exploratory in nature. Often, end-user programmers (non-professional) indulge in opportunistically creating their programs. To facilitate learning and reuse of code, most end-user programming environments provides online repositories. While the provision of programs in repositories helps support end-user programming to an extent, finding and reusing an appropriate program variant is a challenging task. In this paper, we explore the reuse behavior of end-user programmers using Information Foraging Theory. We conducted an empirical study of eight end-user programmers, qualitatively analyzed their information-seeking behavior while reusing program variants, and report new cue types and strategies specific to end-user programmers.**

I. Introduction

Reusing or adapting parts of a program to a current task context is a common phenomenon while programming. This phenomenon is more prevalent with end-user programmers (EUPs), a class of programmers who engages in creative tasks which are exploratory in nature [1]. EUPs are known to create code opportunistically, incrementally, but rarely at one lifecycle phase at a time [2]. Henceforth, EUPs explore and evaluate code alternatives and "glue" components together or modify existing program code to suit a new context or problem [3]. To facilitate reuse, most end-user programming environments provide online repositories.

Online repositories contain programs and its 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]. Furthermore, code variants are created because of the provision of creating code clones or allowing the use of programs as subprograms [11]. Although EUPs may view their programs as "throw-away," their code often ends up being long-lived and, in many cases, is reused by other EUPs [10, 12, 13]. While the provision of programs in repositories helps support end-user programming to an extent, finding and reusing appropriate variants is a challenging task.

Reuse in programming is generally difficult and is more challenging in the context of reusing earlier variants found in online repositories. Variants in close proximity, both spatially and temporally, increase the cognitive effort for both novice and experienced EUPs [35]. Therefore, EUPs forage to localize appropriate program variants and code snippets for reuse and calculate their proximity to the desired program variant as they go. Hence, we used Information Foraging Theory (IFT) to understand the reuse behavior of EUPs.

In IFT, a predator (EUPs) forages for prey (e.g., program or parts of the code) while reusing programs by following cues (e.g., label on links) in patches (e.g., web pages, IDEs). IFT has been studied and applied to (1) the process of "foraging" by web users [14], (2) navigation and debugging of programs by professional programmers [15, 16], (3) EUPs to understand their debugging behavior [17], and (4) novice programmers to understand reuse in variants developed from same project [18].

In this paper, we use IFT to explore the reuse behavior of EUPs in online repositories. We conducted an empirical study of EUPs and qualitatively analyzed the information-seeking behavior. The primary research question is:

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

- *Between-variant foraging*: How do EUPs forage between variants 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?

II. Background and Related work

A. Variations and Variants

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" [18]. Users can access and modify the contents of variants to create a newly modified program in accordance to their desired specifications. For example, a MATLAB project in a directory may be called a variant; after changes have been made to the code, it becomes a new variant for further reuse.

Many variation-supporting tools have been utilized to compare code and create code alternatives [19, 20, 21, 22]. Hartmann et al. introduced the Juxtapose tool to edit code in parallel, quickly create code alternatives, and simultaneously compare code outputs [19]. Karlson et al. introduced the "versionset," a representation of variants from a single source, to help better manage personal information (i.e., personal repository) [23]. Kuttal et al. created tools to allow EUPs to manage a collection of "past and present variants" originating from a single artifact [24, 25].

978-1-5090-0252-8/16/$31.00 ©2016

B. Information Foraging Theory

Pirolli and Card derived information foraging theory (IFT) from optimal foraging theory which models how users, as predators, follow scents emanated from cues to track information: their prey [26]. Cues are information features, such as text associated with hypermedia, 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. Patches are information sources such as the programming environment, file explorers, and web pages found within the variant [18]. Users sift through the cues, determine the strength of the scent, then follow this trail of scent to attain the information they desire. Predators also modify their environment for enrichment [27].

Using IFT, Pirolli et al. examined and explained the behavior of users as they forage for information on the web [14, 28, 29, 30]. By this theory, models were developed to predict user navigation through the web [14, 29]. Software engineering fields such as debugging, code reuse, code maintenance, and navigation behavior modeling and prediction [15, 16, 31, 32, 33] has also benefitted from IFT. Lawrance et al. presented the PFIS2 model which suggests navigation tools can predict appropriate places to guide developers to fix bugs in their code [33]. Piorowski et al. further progressed the PFIS2 model to better understand programmers' goals and strategies as they forage to debug their programs [34]. Kuttal et al., using IFT, reported categories for the cues and strategies EUPs utilized while debugging in mashups [17].

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 with temporally similar variants [18]. Inspired by their work, we utilized their model to understand how EUPs forage for temporally and spatially similar variants in online repositories which contain several artifacts from different developers but share common goals and attributes.

III. Methodology

A. Environment

App Inventor Gallery (AIG): The App Inventor environment is a software development platform designed for mobile Android programs. This gallery contains a database of projects created by a community of 3 million users. As of 2013, these programs numbered 3.2 million for App Inventor Classic and 140,000 for App Inventor 2, the second iteration of App Inventor [7]. The arrows in Fig. 1 show transitions of a user foraging in AIG. The user begins searching for variants in a search bar using keywords and a list of variants is returned. Opening a variant exposes the users to the patches: the designer, the code editor, and the emulator.

Fig. 2. App Inventor Repository with variants and patches.

File Exchange (FE): File Exchange is an online repository maintained by Mathworks. We chose the File Exchange repository because of MATLAB's popularity, a scientific and engineering programming language, which also provided a convenient sampling for our study. Similar to AIG, arrows in Fig. 2 indicate transitions of a user foraging in FE with a difference that a list of patches, in which a user can select, will be returned upon variant selection.

B. Study Design

To qualitatively observe participants' foraging behaviors in-depth, we used a small but more generalizable population by placing the participants[1] in two different environments (four in AIG and four in FE) in the study. Since most EUPs are known to perform opportunistic programming [1], we recruited eight non-Computer Science students from the University of Tulsa. AIG participants had no prior experience and learned the language and environment through the tutorial, while FE participants had 2+ years of experience in MATLAB, thus representing novice and expert EUPs.

After completing a background questionnaire, both groups were asked to use online repositories with keyword-based search engines to accomplish their task. We then conducted a formative lab study using think-aloud protocols [36]. Participants were given 50 minutes to complete two tasks. Two tasks[1] were chosen to observe cues and strategies which participants used while foraging within (Task 1) and between (Task 2) 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. Task 1 aimed to find a project (destination variant) to integrate code to while Task 2 aimed to find another project (source variant) to integrate code from.

During the study, we recorded what the participants said and their on-screen interactions using Morae software. Then we showed recordings of their actions and verbalizations to conduct retrospective interviews. We gained insights into participants' thought processes and barriers while they explored, understood, and selected between- and within-variants. Furthermore, the interviews helped us triangulate our lab findings. Both the study and retrospective interview were administered on an individual basis with an observer.

C. Qualitative Analysis

We segmented the transcripts of participants' think-aloud videos into 30-second segments (multiple cues can be in one segment) and qualitatively analyzed for cues and behaviors. In

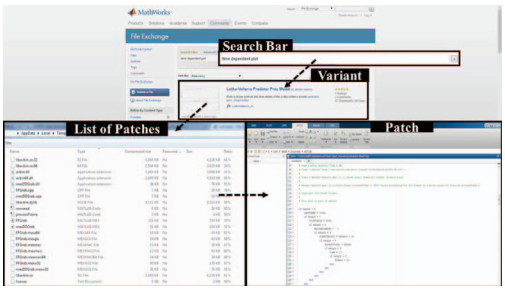

Fig. 1. File Exchange Repository with variants and patches.

order to correctly identify cues/behaviors associated with participants' foraging behaviors, we qualitatively coded the data by first using cues/behaviors established in previous works [18, 37]. For those not found in previous works, we created new cues/behaviors[1] to code the phenomena exhibited by participants in the study. In order 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 [38] and achieved 89% of agreement. After an agreement was made, one researcher coded the rest of the transcripts of participants' verbalization and actions.

IV. Results

In line with the Reuse Model of Variants, cues and behaviors were collected at two different stages: finding and evaluating (1) the current context and (2) the usage context [18].

A. Stage 1: Current Context

In the current context, a predator forages between- and within-variants to find and evaluate a destination variant/patch (where code is imported into) to facilitate their current task.

1) Destination Variant: Find. To find a destination variant (program/project), participants foraged in online repositories using internal (e.g. FE, AIG) or external (e.g. Google) search engines.

***Enrichment Strategies**:* For getting better search results using search engines, our participants used the following enrichment strategies to "mold the environment" [27]:

Orienteering Queries. As similarly described by Teevan et al., participants used the "Orienteering" approach while formulating queries [39]. Especially for first query formulation, participants utilized keywords from the task.

Semantically Similar Keywords in Domain. While foraging for variants using various query formulations, participants recognized the recurrence of semantically similar word(s). They substituted these reoccurring keywords for similar keywords in their queries to get better search results as 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 to use the word 'animation' ... to get better results.*"

External Search to Narrow Internal Search. Participants used External search engines to help them narrow results in the internal search engines as P5 used Google to narrow results in FE using the query "`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. Participants used recency to narrow the results for more relevant variants. For example, P1 mentioned that she explored recent variants because she expected that the browser history would remember the searches made by previous participants in the study. Another participant, P6, noted how old app "*may not be so good*."

Ranking of Results. Rankings, in which variants appeared in the search results, were frequently utilized by the 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 the participants from going to the next page of search results to forage for variants.

2) Destination Variant: Evaluate. To evaluate a destination variant, participants determined whether the variant 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.

Visual. Denote the images and snapshots—usually presented along with a variant's details in a search result—associated with variants. The use of visual cues (images) was mentioned by six participants as a factor to further explore the program. These cues may depict a snapshot of the project, an icon of the program's function, or may be irrelevant or nonexistent in the project. After finding a relevant variant image, P2 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. Title/name given by the developer. All participants mentioned they used titles to guide them and five participants 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 [18].

Description-Inspired. Texts describing the functionality of the program. Six participants used descriptions of the variant or its description page to evaluate program functions. While the detailed description gave a better understanding of the program's functionality some were valued less as P1 remarked the description had *"short descriptions*."

Familiarity. Re-exposure to a project 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. In online repositories, variants were 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.

[1] For details see http://sandeepkuttal.ens.utulsa.edu/notes.pdf

Number of Functions: If the variant had too many or too few functions, some participants abandoned it as P1 noted, "*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 "*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 lamented on not paying more attention to the views and the ratings. The ratings also confused participants as P6 remarked, "*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 that the participant expected the results to be sorted by ratings.

Navigational Strategies:

Depth-First Strategy: Understanding the variants before acting on the results. Some participants stayed on one variant at a time to understand and evaluate before going to the next.

Breadth-First Strategy: Acting on variants before seeking information. Before further understanding the variants, some participants opened multiple variants in new tabs as a list of variants to evaluate. We found participants evaluated all the variants before choosing which patch to open or they evaluated each variant and then opened/downloaded patches regardless of how many variants were opened.

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

3) Destination Patch: Find and Evaluate. In a variant, participants found patches to modify the program (refer to Fig 1. and 2.). Each patch found were then evaluated based on the cues (similar to [24, 37]). Output-inspired cue was the most popular (131 instances) cue used to gather information. Code-inspired cues (72 instances) were used by participants as they read the code *"line by line"* to understand it, according to P5. In the current context, internal documentation cues (36 instances), namely tool tips and commentary, were used to understand the patch. Few participants (7 instances) used code-status-inspired cues which describe/display non-invasive errors in code. These cues are elusive and hard to find [18]. The only strategy used while finding a destination patch by participants was keeping trails of the patches. After evaluating patches, some participants kept trails of patches by keeping the patches or folders holding the patches open.

B. Stage 2: Usage Context

In the usage context, a predator forages between- and within-variants to find and evaluate a source variant/patch (where code is integrated from) to facilitate their current task.

1) Source Variant: Find and Evaluate. After finding and evaluating the destination patch (in Stage 1), participants exhibited different behaviors to find a source variant: some stuck to the destination variant they chose while others actively searched and evaluated other source variants. Participants who searched and evaluated for source variants used similar strategies and cues to find and evaluate the source variant as they did in destination variant on Stage 1.

2) Source Patch: Find and Evaluate. Participants utilized similar strategies to find and evaluate source patches as in Stage 1. However, participants searched for different source patches, mostly external documentation such as: forums, Stack Overflow, MATLAB Answers, wikis, tutorials, and etc. While evaluating patches in usage context, our participants used the same cues they used in the current context with the addition of External Documentation-Inspired cues. Participants used external documentation to evaluate if a patch had desirable resources (e.g., code snippets).

V. Discussions

Our research is first to study EUPs' reuse behavior in online repositories using IFT constructs. Unlike Ragavan et al.'s study, where variants were associated with Timestamps (cues), our study using online repositories had more cues associated with the variants. Thus, our participants used cost-benefit analysis using cues, with preference to social aspect and familiarity, to evaluate variants and patches while foraging. The study also helped us envision tool designs:

Removing/Adding Search Trails: Our participants mentally or physically kept a trail while comparing across variants and patches. This suggests a need for tools to support such behaviors and help reduce not only the cognitive load of EUPs but also the time they spent while foraging for similar variants.

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, that may facilitate EUPs to just recognize the variants. Some of the search engines like specification-based searches [42], CODEBROKER [41], or behavior-based clustering for visual programs [40] can be helpful to an extent.

Automatic Cues Extractions: Our results suggest that online repositories as well as IDEs should facilitate the automatic extraction of the program outputs to give improved visual cues and textual descriptions of a program. These cues should be associated with the variants to better facilitate evaluation of variants and patches.

Allowing Explorations: Supporting recommendation systems for variants or having search engines integrated into IDEs provides easy access to relevant variants and help facilitate reuse.

Overall, we report new cues and strategies not yet reported in IFT literature, which provide implications for researchers and practitioners and help design better tools for accessing online repositories. Our study also opens questions regarding search engines: Is keyword-based searching for programs and their variants enough?

VI. REFERENCES

[1] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer, "Opportunistic programming: Writing code to prototype, ideate, and discover," in IEEE Software, vol. 26, no. 5, 2009, pp. 18–24.

[2] M. Burnett and B. Myers, "Future of end-user software engineering: beyond the silos." in ICSE Companion Proceedings, 2014, pp. 201-211.

[3] R. Holmes and R. J. Walker, "Systematizing pragmatic software reuse," ACM Trans. Softw. Eng. Methodol. 21, 4, Article 20, Feb 2013, 44 pages.

[4] File Exchange: http://www.mathworks.com/matlabcentral/fileexchange/

[5] MATLAB: http://www.mathworks.com/products/matlab/?requestedDomain=www.mathworks.com

[6] App Inventor Gallery online repository: http://appinventor.mit.edu/explore/blogs/shay/2013/04/join-app-inventor-community-gallery.html.

[7] App Inventor: http://news.mit.edu/2013/app-inventor-launches-second-iteration.

[8] Scratch Repository: https://scratch.mit.edu/explore/?date=this_month

[9] Scratch Editor: https://scratch.mit.edu/projects/editor/?tip_bar=home

[10] K. T. Stolee, S. Elbaum, and A. Sarma, "Discovering how end-user programmers and their communities use public repositories: A study on Yahoo! Pipes," in Information & Software Technology, vol. 55, 2013, pp. 1289–1303

[11] S. K. Kuttal, A. Sarma, and G. Rothermel, "Debugging support for end-user mashup programming," in CHI, 2013, pp. 1609–1618.

[12] C. Bogart, M. Burnett, A. Cypher, and C. Scaffidi, "End-user programming in the wild: A field study of coscripter scripts," in 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] W. Fu, and P. Pirolli. "SNIF-ACT: a cognitive model of user navigation on the world wide web," in Human–Computer Interaction vol. 22, no. 4, 2007, pp. 355-412.

[15] N. Niu, A. Mahmoud, and G. Bradshaw, "Information foraging as a foundation for code navigation (NIER track)," in ICSE, 2011, pp. 816-819.

[16] J. Lawrance, M. Burnett, R. Bellamy, C. Bogart, and C. Swart, "Reactive information foraging for evolving goals," in CHI, 2010, pp. 25-34.

[17] S. K. Kuttal, A. Sarma, and G. Rothermel, "Predator behavior in the wild web world of bugs: an information foraging theory perspective," in VL/HCC, 2013, pp. 59-66.

[18] S. S. Ragavan et al., "Foraging among an overabundance of similar variants," in CHI, 2016, pp. 3509-3521.

[19] B. Hartmann, S. Follmer, A. Ricciardi, T. Cardenas, and S. R. Klemmer. "d.note: revising user interfaces through change tracking, annotations, and alternatives," in CHI, 2010, pp. 493-502.

[20] R. Kumar, J. O. Talton, S. Ahmad, and S. R. Klemmer, "Bricolage: example-based retargeting for web design," in CHI, 2011, pp. 2197-2206.

[21] M. Terry and E. D. Mynatt, "Side views: persistent, on-demand previews for open-ended tasks," in UIST, 2002, pp. 71-80.

[22] M. Terry, E. D. Mynatt, K. Nakakoji, and Y. Yamamoto "Variation in element and action: supporting simultaneous development of alternative solutions," in CHI, 2004, pp. 711-718.

[23] A. K. Karlson, G. Smith, and B. Lee, "Which version is this?: improving the desktop experience within a copy-aware computing ecosystem," in CHI, 2011, pp. 2669-2678.

[24] S. K. Kuttal, "Leveraging variation management to enhance end users' programming experience," in *ETD Collection for University of Nebraska – Lincoln*, Ph.D. dissertation, CS, UNL, Lincoln, NE, 2014.

[25] S. K. Kuttal, A. Sarma, and G. Rothermel, "On the benefits of providing versioning support for end users: an empirical study," in *ACM Transactions on Software Engineering and Methodology,* vol. 21, no. 2(9), 2014.

[26] P. Pirolli and S. Card, "Information foraging in information access environments," in CHI, 1995.

[27] P. Pirolli, "Rational analyses of information foraging on the web," in *Cognitive science,* vol. 29, no. 3, 2005, pp. 343-373.

[28] P. Pirolli, W. Fu, E. Chi, and A. Farahat, "Information scent and web navigation: Theory, models and automated usability evaluation," in *Proceedings of HCI International*, 2005.

[29] P. Pirolli and 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.

[30] P. Pirolli. 1997, "Computational models of information scent-following in a very large browsable text collection," in CHI, 1997, pp. 3-10.

[31] S. D. Fleming et al., "An information foraging theory perspective on tools for debugging, refactoring, and reuse tasks," ACM Transactions on Software Engineering and Methodology, vol. 22, no. 2(14), 2013.

[32] J. Lawrance, C. Bogart, M. Burnett, R. Bellamy, and K. Rector, "How people debug, revisited: an information foraging theory perspective," in VL/HCC, 2009, pp. 117-124.

[33] J. Lawrance, R. Bellamy, M. Burnett, and K. Rector., "Using information scent to model the dynamic foraging behavior of programmers in maintenance tasks," in CHI, 2008, pp. 1323-1332.

[34] D. Piorkowski *et al.*, "Modeling programmer navigation: a head-to-head empirical evaluation of predictive models," in VL/HCC, 2011, pp. 18-22.

[35] K. Martzoukou, "A review of Web information seeking research: considerations of method and foci of interest." *Information Research,* vol. 10, no. 2, 2005.

[36] C. H. Lewis, "Using the 'Thinking Aloud' method in cognitive interface design," RC 9265, IBM, 1982.

[37] D. Piorkowski et al., "To fix or to learn? How production bias affects developers' information foraging during debugging," in ICSME, 2015.

[38] P. Jaccard. "Étude comparative de la distribution florale dans une portion des Alpes et des Jura," Bulletin del la Société Vaudoise des Sciences Naturelles, vol. 37, pp. 547–579, 1901.

[39] J. Teevan, C. Alvarado, M. S. Ackerman, and D. R. Karger, "The perfect search engine is not enough: A study of orienteering behavior in directed search," in CHI, 2004, pp. 415–422.

[40] S. Surisetty, C. Law and C. Scaffidi, "Behavior-based clustering of visual code," in VL/HCC, 2015, pp. 261-269.

[41] Y. Ye and G. Fischer. *"Supporting reuse by delivering task-relevant and personalized information,"* in ICSE, 2002, pp. 513-523.

[42] K.T. Stolee and S. Elbaum, "Toward semantic search via SMT solver," in FSE, Art. 25, 2012