Chapter

How Do Web-Active End-User Programmers Forage?

Sandeep Kaur Kuttal, Abim Sedhain and Benjamin Riethmeier

Abstract

Web-active end-user programmers spend substantial time and cognitive effort seeking information while debugging web mashups, which are platforms for creating web applications by combining data and functionality from two or more different sources. The debugging on these platforms is challenging as end user programmers need to forage within the mashup environment to find bugs and on the web to forage for the solution to those bugs. To understand the foraging behavior of end-user programmers when debugging, we used information forging theory. Information foraging theory helps understand how users forage for information and has been successfully used to understand and model user behavior when foraging through documents, the web, user interfaces, and programming environments. Through the lens of information foraging theory, we analyzed the data from a controlled lab study of eight web-active end-user programmers. The programmers completed two debugging tasks using the Yahoo! Pipes web mashup environment. On analyzing the data, we identified three types of cues: clear, fuzzy, and elusive. Clear cues helped participants to find and fix bugs with ease while fuzzy and elusive cues led to useless foraging. We also identified the strategies used by the participants when finding and fixing bugs. Our results give us a better understanding of the programming behavior of web-active end-users and can inform researchers and professionals how to create better support for the debugging process. Further, this study methodology can be adapted by researchers to understand other aspects of programming such as implementing, reusing, and maintaining code.

Keywords: Information Foraging Theory, End-user programming, Debugging, Visual Programming, Web Mashups

1. Introduction

In modern times, mass communication, mass media, and networking technologies have enabled access to vast amounts of knowledge that are distributed across many continents and time-zones, thus allowing web-active end-users to achieve great feats.

Web-active end-users (also referred to as end-users or end-user programmers) are people who lack programming experience but are engaged in internet activities [1]. There is a substantial number of web-active end-users and their number is continuously growing. The end-users often create applications to complete tasks such as finding apartments to rent in a certain location, tracking flights, and alerting

drivers regarding traffic jams. One approach to create such applications is utilizing web mashups programming environments.

Web mashup programming environments allow for creating applications from distributed heterogeneous web sources and functions. Most of the mashup programming environments are visual in nature. Some examples include Yahoo! Pipes [2], IBM mashup maker [3], xfruit [4], Apatar [5], Deri pipes [6], and JackBe [7]. The visual nature of these programming environments allows application creation using code abstraction to ease the programming process. However, the abstraction of code can add complexity of accessing the information, debugging, and comprehending large programs within these environments [1, 8, 9].

Further, end-users create mashup applications by seeking information from the complex ecosystem of the web, which is composed of evolving heterogeneous formats, services, standards, and languages [8]. Seeking information on the web is challenging, as the relevant information is scattered across numerous web sources that end-users must find and manually analyze, an information-seeking problem that costs both time and cognitive effort.

In this chapter, we observe the behavior of end-users while debugging, one of the most difficult aspects of programming [10]. Debugging mashup programs is even more challenging as end-user programmers must locate bugs within the abstract web mashup environment and then locate solutions on the web to fix bugs. The lack of debugging support within mashup environments increases the complexity of finding bugs [9]. Further, finding correct solutions to fix bugs is complicated as the web is a huge compilation of heterogeneous resources.

Currently, it is not clear how web-active end-users seek for bugs in their program and their solutions on the web. Hence, we used an information seeking theory called Information Foraging Theory.

Information Foraging Theory (IFT) can expand our understanding of the information-seeking problems of web-active end-user programmers while debugging. IFT posits that people seek information in the same manner as predators forage for their prey, where predators are the end-users, and the prey is the bugs or bug fixes they are searching for. The hunting grounds or 'patches' where web-active end-users search for these bugs or fixes would be their IDE or the websites they visit and the scents the web-active end-users follow are given by different cues (e.g., links) found on the web [11–15]. IFT has been applied successfully to diverse domains such as documents, the web, user interfaces, and programming environments [15–23].

Past research on web mashups have focused on creating web tools that increase the ease and effectiveness of creating applications by end-user programmers [24–28]. While past IFT research on programming environments has investigated debugging and navigational behavior of professional programmers [19–21]. No prior research exists to understand the debugging behavior of web-active end-user programmers. The only research relevant to this chapter is our own [8], where we created a debugging support for web mashups and investigated the debugging behavior of end-user programmers using IFT with and without the support. Based on this prior research, we found IFT to be the most relevant choice to understand the information-seeking behavior during mashup debugging.

To understand the debugging behavior of end-user programmers we conducted a controlled lab study of eight students who were not computer science majors. The study participants completed their tasks using Yahoo! Pipes, a mashup environment, as it provided the best debugging support at the time. The participants completed two debugging tasks using a think-aloud protocol. We investigated how end-users forage for information within the IDE as well as the web using IFT

theory. Our analyses discovered new cues and strategies that end-user programmers pursued while locating the bugs in the mashup environment and foraging the web for fixing the bugs.

This chapter is organized as follows. Section 2 describes the debugging behavior of end-user programmers. Section 3 describes Information Foraging Theory, IFT terminologies from Yahoo! Pipes, and relevant literature. Section 4 describes the background and related work on web mashups, and Yahoo! Pipes. Section 5 describes the methodology and results from the lab study. This section discusses the cues utilized by end-user programmers and their behavior during debugging tasks and provides recommendations. Section 6 summarizes our findings and suggests how web mashup environments can improve the debugging process.

2. Debugging and end-user programmers

Debugging is the process of finding and fixing bugs in the code. Programmers often struggle to debug and hypothesize the "when", "why" and "how" of the bug [29–32]. Debugging is even more challenging for end-user programmers as in one study [33] they spent two-thirds of their time foraging for bugs, while professionals spent only half of their time.

Professionals and end-users use web resources to complete their programming tasks. For example, in one study, novice programmers spent about 19% of their programming time in foraging the web for information such as selecting and using tutorials, searching with synonyms, finding code snippets, and using the web to debug [34], while they spent 35% of their time navigating source code [35]. Vessey [36] investigated both professionals and end-users' debugging approach and found that professionals took a breadth-first approach whereas end-users took a depth-first approach. Our study found that in mashup environments the end user programmers struggle foraging for solutions to bugs on the web.

A major huddle for programmers during debugging is understanding the error messages to fix bugs in the code. Naveed and Sarim [37] analyzed how presentation of error messages affected debugging and programming in IDEs. To fix a bug, first programmers must understand what the error is and where it is located. Mashup environments tend to show errors without much explanation or direction for the end-user to comprehend [9]. End-users struggle to adapt code from tutorials and web forums [38] while fixing bugs. They often struggle with debugging due to lack of knowledge and experience in software engineering and interactive programming environments [39]. Our study confirms that end-user programmers struggle with the lack of or unclear error messages in IDEs.

Understanding end-user programmers' behavior while debugging can help to build better debugging tools that facilitates programming tasks effectively and efficiently. Phalgune et al. [40] studied oracle mistakes - mistakes users make about which values are right and which are wrong - that impact the effectiveness of interactions, testing, and debugging support for end-users. Kuttal et al. [41] added version support to Yahoo! Pipes and investigated how versioning can help end-user programmers to create and debug mashups. Servant et al. [42] create support that allowed panning and zooming of a canvas that contained the snapshots of the code. Myers and Ko suggested various interaction features for IDE to improve debugging such as full visibility of code and timeline visualization of changing values of variables at run-time [43]. Our study helps to understand how end-user programmers debug from a theory perspective that can inform better debugging support for mashup environments.

3. Web mashups

Web mashups allow end-users to build applications by integrating data and functionalities from various web services into a single application. The visual web mashup programming environments facilitate easy creation of applications by end-user programmers who have very little knowledge and experience in programming. Mashup environments provide a full set of functions to the end-users to build new applications.

End-users often create situational mashups as per their specifications [44]. For example, a mashup can take data from Instagram and combine it with Google Maps to display the most recent images and videos of any given location. Users can get the data from APIs, Information Feeds (e.g., Really Simple Syndication (RSS)), or they can collect data by scraping various web pages. Mashup application can be executed within the client's browser, in a server, or combination of both. The advantage of rendering the application in a client's web browser is to give users the opportunity to interact with it. Mashups are popular because of their dynamic content creation and ability to build and share applications through publicly hosted repositories [45].

End-users often develop mashup applications using visual black-box oriented programming environments. Mashup programming environments such as Yahoo! Pipes [2], IBM mashup maker [3], xfruit [4], Apatar [5], Deri Pipes [6], and JackBe [7] provide an easy-to-use visual environment to support the mashup development. Cappiello at el. [46] researched mashup development frameworks oriented towards end-user development to allow users to compose different resources at different levels of granularity relying on the user interface (UI) of the application. Ennals and Gay created MashMaker [24], a tool which allowed end-users to create web mashups without needing to write much code/script. Other mashup creation tools to facilitate end user programmers include MapCruncher [25], Marmite [26], Automator [27], Creo [28], and TreeSheet [47]. Rather than directly studying mashup environments or creating new mashup tools, we qualitatively observe how end-users debug and forage for solutions in programs built in these mashups.

Grammel and Storey [9] investigated various mashup development environments and found lack of debugging support in these environments. Similarly, Stolee and Elbaum [48] studied how we can improve the refactoring of pipe-like mashups, i.e., Yahoo! Pipes for end-users. We focus on understanding end-user programmers' behavior while debugging mashups instead of creating support for mashups.

3.1 Yahoo! Pipes

Now defunct, Yahoo! Pipes was introduced in 2007 and was one of the most popular mashup creation environments that helped users to "rewrite the web" during its existence. During its first year of existence, the Yahoo! Pipes platform executed over 5,000,000 pipes per day. As a visual programming environment, Yahoo! Pipes was well suited for representing the solutions to dataflow-based processing problems. Yahoo! Pipes "programs" helped in combining simple commands together such that the output of one acted as the input for the other. The Yahoo! Pipes engine also facilitated the wiring of modules together and the transfer of data between them.

The Yahoo! Pipes environment was made up of three major components: the canvas, the library (list of modules), and the debugger (refer **Figure 1**). Users used the canvas to create the pipes. The library situated to the left of the canvas, consisted of various modules that were categorized according to functionality. Users dragged modules from the library and placed them on the canvas, then proceeded to connect them to other modules as their need. The debugger, located at the bottom, helped users check the runtime output of the modules.

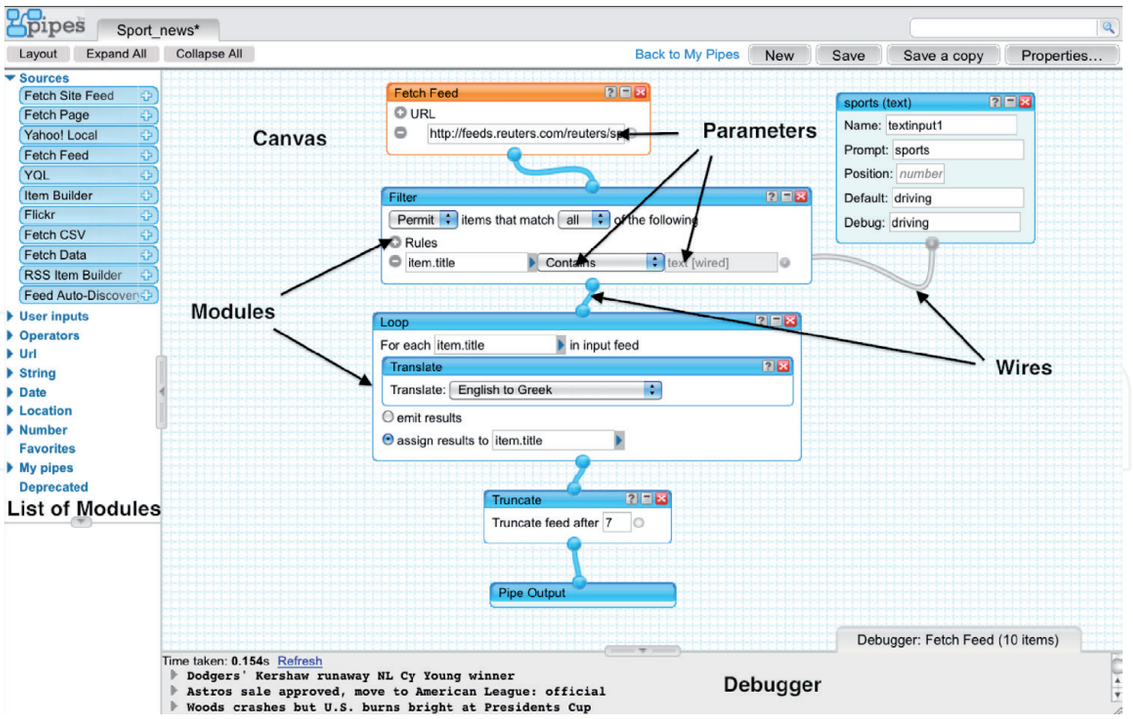

Figure 1.
Yahoo! Pipes.

The inputs and output of the pipes supported different formats. For input, most common formats were APIs, HTML, XML, JSON, RDF, and RSS feeds. Similarly, pipe output formats were RSS, JSON, and KML. The inputs and outputs between modules were primarily RSS feed items consisting of parameters and descriptions. Yahoo! Pipes modules provided manipulation actions that could be executed on these RSS feed parameters. In addition to items, Yahoo! Pipes also allowed datatypes like URL, location, text, number, and date-time to be defined by users.

Figure 1 shows the interface and components of the Yahoo! Pipes environment. The pipe displayed in the figure takes Reuter's Newsfeed (RSS feed) as input using a Fetch Feed module which is then filtered (using a Filter module) based on users' input (sports). These results are converted from English to Greek using a Translate module inside a Loop module. The pipe titles are limited to the first seven results using the truncate module. In **Figure 1**, the debugger window displays the runtime output from the Fetch Feed module.

Yahoo! Pipes allowed the creation and rendering of the pipes on the client side while the executing and storing of the pipe was done on the Yahoo! Servers. The data between the client and server was transfer using JSON format. Yahoo! Pipes allowed end-users to share their pipe (code) as well as reuse other user's pipes by cloning.

Stolee et al. [49] analyzed 32,000 mashups from Yahoo! Pipes repositories based on popularity, configurability, complexity, and diversity. Wang and Wang [50] used Yahoo! Pipes to build a mobile news aggregator application. We used Yahoo! Pipes for this study as it had the best debugging support at the time of the research.

4. Information Foraging Theory and Yahoo! Pipes

Information Foraging Theory (IFT) was developed by Pirolli and Card [11] to understand how people search for information. IFT was inspired by optimal foraging theory, which is a biological theory explaining how predators hunt for

their prey in the wild. Optimal foraging theory predicts whether a prey (animal) will try to maximize the energy it gains or minimize the expense to obtain a fixed amount of energy [12]. Similarly, while foraging for information, users must realize their maximum return on information gain at minimum expenditure of their time. Therefore, users, when possible, will modify their strategies to maximize their rate of gaining valuable information [13]. **Table 1** elaborates the IFT terminologies along with examples from Yahoo! Pipes.

IFT has helped to improve the understanding of the users' behaviors and interactions on the web. In the very beginning, research was done for general Internet users, which led to the foundation of IFT [15, 18, 51]. Research has been done to observe and study foragers on the web [8, 15, 21, 51]. IFT has been used to improve the usability of web sites [52] as it has helped to explain and predict why people click a particular link, text, or button on a website [14]. In this research, we qualitatively analyze multiple end-user's foraging behavior to find solutions for their bugs on the web.

IFT has also been used to understand software engineering and software development [8, 19, 20] along with its collaborative environments [17]. Piorkowski et al. have explored foraging behavior and the difference in foraging between desktop and mobile integrated development environment (IDE) [53]. Niu et al. used IFT to design navigation affordances in IDEs [54]. Similarly, IFT has been used to find out the optimal team size for open-source projects [55]. IFT can help to understand the foraging behavior of web-active end-user programmers when engaged in programming activities such as comprehension, reusage of code, implementation, debugging and testing. This research focuses on the debugging behavior of web-active end-user programmers.

Researchers have built computational models of user information foraging behavior when completing tasks [14, 56, 57]. These models have also helped in predicting the effects of social influences on IFT [58]. The researchers have developed

IFT Terminologies	Definitions	Bug Finding (Examples)	Bug Fixing (Examples)
Prey	Bugs; solutions	Finding bug B2 (url does not lead to the right web site) in Fetch Feed module	Finding the correct url and putting it in the Fetch Feed module that contains B2
Information Patch	Localities in the code, documents, examples, web-pages and displays that may contain the prey [23]	Yahoo! Pipes Editor, help documents, help examples	Web pages
Information Feature	Words, links, error messages, or highlighted objects that suggest scent relative to prey	API Key Missing error message "Error fetching [url]. Response: Not found (404)" for bug B1	Finding the right API key from the website
Cues	Proximal links to patches	"about this module" link to the example code related to specific module	"Key" link to the Flickr page to collect the API key
Navigate	Navigation by users through patches	To find bug B2 the user navigated through Yahoo! Pipes editor to external web site	To correct bug B2 participant navigated to various web sites to find the required url

Table 1.
IFT Terminologies from the Yahoo! Pipes Perspective [2].

the WUFIS model for the web [6] and the PFIS model for programmers foraging in IDEs [19, 20]. Ragavan et al. analyzed the novice programmers' foraging in the presence of program variants [22] and built a predictive model [59] inspired by the PFIS model [23, 60]. Our focus is to understand the end-user foraging behavior before creating such computational models.

5. Understanding debugging behavior using an information foraging theory perspective

To understand how end-user programmers forage mashup IDEs (Yahoo! Pipes) for finding bugs and the web for finding solutions for the bugs, we conducted a controlled lab study.

5.1 Lab study using Yahoo! Pipes

Our study observed eight university students who had no background in computer science but had experience with one web language. The students were from diverse fields such as engineering, finance, mathematics, and natural sciences. The participants completed the background questionnaire, a short tutorial on Yahoo! Pipes, and a pilot task to practice programming with Yahoo! Pipes. Once the participants felt comfortable with the Yahoo! Pipes environment, they completed two tasks using the think-aloud method.

The participants were given Yahoo! Pipes programs that were seeded with bugs. The first task (Yahoo! Pipes Error) was a pipe program that was seeded with bugs detected by Yahoo! Pipes and displayed a relevant error message. The second task (Silent Error) was seeded with bugs that were not detected by Yahoo! Pipes and therefore did not display an error message. Further, both tasks contained two classes: top level and nested. Top level contained bugs that were easy to comprehend while the nested class contained sub-pipes with bugs. These sub-pipes needed to be opened in a separate IDE to be found. The details of the tasks can be found in **Table 2**.

Participants' verbalization and actions were transcribed and analyzed using IFT theory. When analyzing the transcripts, we found various cues and strategies used by our participants.

5.2 Types of cues followed by end-user programmers

In finding the bugs and their fixes, participants followed cues. Based on the strength of the cues, they can be classified as clear, fuzzy, and elusive. *Clear cues*

Task	Class	Bugs	Details
Yahoo! Pipes Error	Top Level	B1	API key missing
		B2	Website not found
	Nested	B3	Website not found
Silent Error	Top Level	B4	Website contents changed
		B5	Parameter missing
	Nested	B6	Parameter missing

Table 2.
Details on seeded bugs in the tasks [2].

helped the forager the most as they were easy to understand and provided a direct link to the bugs or their fixes. Hence, they were less costly as they helped participants to spend less time finding and fixing the bugs. *Fuzzy cues* did not have complete information that could lead to a bug. Hence, these cues either lead or mislead to a valuable patch containing prey and were somewhat costly in terms of time spent. *Elusive cues* were very difficult to locate due to absence of direct links to the bugs. These cues were the costliest, as participants often wasted their time foraging for prey in useless patches.

Cues	Description	Example
Clear Cues	Cues that were clear and easy to understand	'*API Key Missing*' cue helped participants look for modules that it was associated with.
Fuzzy Cues	Cues that were difficult to understand	'*org.xml.sax.SAXParseException*' cue was hard for participants to understand as they didn't know what it meant.
Elusive Cues	Cues that were difficult to find	This cue was shown when a fault was nested.

5.3 Debugging behavior of end-user programmers

Participants foraged Yahoo! Pipes IDE to find the bugs and the web to fix the bugs. **Table 3** shows the number of bugs located and fixed by each participant. The results show that end user programmers struggled to debug their pipe programs. The key findings were:

5.3.1 Locating and fixing Yahoo! errors was easier than "silent errors"

The Yahoo! Errors B1 and B2 were easily located by the participants (refer **Table 3**). Yahoo! errors supported clear cues as these bugs had detailed error messages from Yahoo! Pipes. As discussed before, the Yahoo! Pipes environment

Participants	Yahoo! Pipes						Silent Errors					
	B1		B2		B3		B4		B5		B6	
	L	F	L	F	L	F	L	F	L	F	L	F
P1	1	1	—	—	—	—	—	—	1	—	—	—
P2	1	1	1	—	—	—	—	—	—	—	—	—
P3*	1	1	1	1	—	—	1	1	1	1	—	—
P4	1	1	1	—	—	—	1	—	1	1	—	—
P5	1	1	1	—	1	—	1	—	—	—	—	—
P6	1	1	1	—	1	1	1	—	1	—	1	—
P7	1	1	1	—	1	1	1	—	—	—	—	—
P8	1	—	1	—	—	—	1	—	—	—	—	—
Total	8	7	7	1	2	1	6	1	4	2	1	0

** represents a participant with prior knowledge of Yahoo! Pipes.*

Table 3.
Bugs Finding and Fixed per Control Group Participant [2].

provides little support for debugging i.e., just observing the output in the debugger window, hence silent errors B4 and B5 were harder for participants to locate and fix. Hence, end-users' programming IDE should support clear cues i.e., displaying and visualizing of the error messages for the programmers.

5.3.2 Locating bugs was easier than fixing bugs

Locating bugs was easier, especially in the presence of clear cues as well as when participants foraged in the restricted single patch of Yahoo! IDE to locate bugs. But when participants had to fix the bugs, they spent a tremendous amount of time foraging through different web pages (multiple patches). Participants used an enrichment strategy of searching on the web to find the valuable patches. But the quality of their search results depended upon the relevance of keywords. Hence, explicitly stating or automating support of the diet constraints (keywords related to bugs) in the search engines can increase the relevance of the results.

5.3.3 Difficult to locate nested bugs, particularly "silent errors"

The nested bugs were the hardest to locate by the participants as they were elusive. In the case of bug B3, three participants were able to find them as they were *clear cues* with error messages that were returned in the pipe output. To detect the silent errors, participants had to systematically analyze each module of the pipe program and check the debugging window. As a result, only one participant was able to locate the B6 bug. Hence, the IDEs should strengthen the cues by making prey/bugs more visible to the programmers through clear cues.

5.4 Strategies while finding Bugs

Participants foraged for finding the bugs using *Hunting*, *Enrichment*, and *Navigation* strategies within Yahoo! Pipes IDE.

5.4.1 Hunting strategy

These strategies reflect how the participants hunted for their prey (bugs). The participants had salient goals and they chose cues based on their prominence. For example, they looked for cues in the output of the pipe program. Most participants pursued the first available cue in the output. This explains why most participants pursued bug B1 and B4 (**Table 3**). The participants were mostly unsuccessful in finding the majority of bugs as participants were persistent and pursued a single bug until they found a fault (depth-first search). The hunting strategies were prompted by the environment itself. Hence, designing environments that facilitate problem solving strategies (such as "sleep on the problem") and make prey more visible can facilitate effective hunting strategies by end-user programmers.

5.4.2 Enrichment strategy

To make prey (bugs) more visible as well as to understand the patch, the participants used various enrichment strategies. They realigned/regrouped the modules so that the connections between them were more visible. For exploring the cues, they kept two patches side-by-side. For example, participants placed the editor and documentation side-by-side for better view of each window. This suggests that IDEs should allow multi-context views allowing end user programmers to view different dimensions of code and allow easy manipulation of the environment.

5.4.3 Navigational strategy

The participants carved out regions based on the data flow structure of Yahoo! Pipes and foraged for cues down each path separately. Whenever they found a weak scent (perceived value), they backtracked and returned to the previous cue or patch. Participants often needed to backtrack for small changes, and this suggests supporting fine-grained backtracking that allows non-linear explorations of past programming history [8, 41].

5.5 Strategies followed when fixing bugs

While fixing the bug, participants used *Enrichment, Navigation* and *Verification* strategies.

5.5.1 Enrichment strategy

Participants searched for all possible cues that led them to fixes for the bugs and aggregated them. Most participants used Google to find the solution for bug fixes. They temporarily collected information to reduce cognitive efforts. For example, participants copied original URLs into the notepad and then started making changes to the pipe programs. Hence, supporting to-do lists can help end-user programmers to complete their tasks systematically [61]. Participants also kept the documents (web document and IDE) open side-by-side like when they searched for bugs, necessitating support for multi-contextual views for code and relevant web pages.

5.5.2 Navigational strategy

The participants skimmed through patches for stronger scents. They used already visited patches as negative evidence in their foraging pursuits. For example, participants closed the web pages immediately when they realized they had already visited them. This prompted the participants to backtrack often to previous cues or patches as they were no longer foraging in the right directions. This suggests the need of tools that allow backtracking across multiple patches.

5.5.3 Verification strategy

After fixing the bugs, participants verified it by rerunning the pipe programs and comparing the output to the given solution (oracle). Verification is a very important step in software engineering and building automated techniques to support verification for end-user programmers can help them produce better quality software applications.

6. Conclusions

Our analysis of the debugging behavior of eight end-user participants using information foraging theory suggests that clear cues were the most cost-effective method for finding bugs in mashup environments. Clear cues created stronger perceived value and helped more in the debugging process allowing end-user programmers to locate bugs more easily when compared to fuzzy or elusive cues. Fuzzy and elusive cues resulted in a hindered debugging progress as end-users would end up in useless patches. In addition, the presence of sub-pipes added additional complexity

to the debugging process as participants were unsure where cues were coming from, even if they were clear. Our study also examined how the participants followed the cues to find solutions to the present bugs.

The participants used three main strategies to locate bugs: hunting, navigation, and enrichment. While hunting they used a depth-first strategy resulting in a persistent pursuit of a single bug. When navigating the participants would use the dataflow structure of the program to perceive the value of the bug's location and would backtrack through relevant program histories to locate the bug. Finally, when using the enrichment strategy, participants would organize their environment by placing their IDE side by side with a web browser or by rearranging the code for easier foraging.

The presence of relevant error messages made these strategies for finding bugs more effective; however, when fixing the bugs by foraging the web different strategies were needed in the absence of clear cues. The participants made use of enrichment, navigation, and verification strategies for fixing bugs. They enriched their patches by finding relevant information through Google, storing URLs of useful websites, and by having these resources open side by side next to the editor. The participants navigated the web and used negative evidence to avoid already visited webpages or unhelpful resources. Then by running the program after implementing fixes, the participants would verify that their solutions fixed the bugs.

Our results suggest mashup programming environments need to facilitate clear clues and support hunting, enrichment, navigational, and verification strategies to facilitate the debugging process for end-user programmers.

Author details

Sandeep Kaur Kuttal*, Abim Sedhain and Benjamin Riethmeier
University of Tulsa, Tulsa, OK, USA

*Address all correspondence to: sandeep-kuttal@utulsa.edu

References

[1] Zang N, Rosson MB. What's in a mashup? And why? Studying the perceptions of web-active end users. In: 2008 IEEE Symposium on Visual Languages and Human-Centric Computing 2008 Sep 15 (pp. 31-38). IEEE.

[2] Yahoo! Pipes. [cited 2015May]. Available from: http://pipes.yahoo.com/pipes/

[3] IBM Mashup Maker. [cited 2015May]. Available from: http://www.ibm.com/software/info/mashup-center/

[4] WMaker. [cited 2021Apr8]. Available from: http://www.xfruits.com/

[5] Apatar - Open Source Data Integration & ETL - Apatar - Open Source Data Integration and ETL [Internet]. Apatar Mashup Data Integration. [cited 2021Apr8]. Available from: http://apatar.com/

[6] Deri Pipes. [cited 2015May]. Available from: http://pipes.deri.org/

[7] Jackbe. [cited 2021Apr8]. Available from: https://jackbe.com/

[8] Kuttal SK, Sarma A, Burnett M, Rothermel G, Koeppe I, Shepherd B. How end-user programmers debug visual web-based programs: An information foraging theory perspective. Journal of Computer Languages. 2019 Aug 1; 53y22-37.

[9] Grammel L, Storey MA. A survey of mashup development environments. In: The smart internet 2010 (pp. 137-151). Springer, Berlin, Heidelberg.

[10] Gould JD. Some psychological evidence on how people debug computer programs. International Journal of Man-Machine Studies. 1975 Mar 1;7(2):151-82.

[11] Pirolli P, Card S. Information foraging in information access environments. In: Proceedings of the SIGCHI conference on Human factors in computing systems 1995 May 1 (pp. 51-58)

[12] Kie JG. Optimal foraging and risk of predation: effects on behavior and social structure in ungulates. Journal of Mammalogy. 1999 Dec 6;80(4):1114-29.

[13] Pirolli P, Card S. Information foraging. Psychological review. 1999 Oct;106(4):643.

[14] Chi EH, Pirolli P, Chen K, Pitkow J. Using information scent to model user information needs and actions and the Web. In: Proceedings of the SIGCHI conference on Human factors in computing systems 2001 Mar 1 (pp. 490-497).

[15] Pirolli P., Fu WT. (2003) SNIF-ACT: A Model of Information Foraging on the World Wide Web. In: Brusilovsky P., Corbett A., de Rosis F. (eds) User Modeling 2003. UM 2003. Lecture Notes in Computer Science, vol 2702. Springer, Berlin, Heidelberg.

[16] Burnett MM. Information Foraging Theory in Software Maintenance. OREGON STATE UNIV CORVALLIS; 2012 Sep 30.

[17] Kwan I, Fleming SD, Piorkowski D. Information Foraging Theory for Collaborative Software Development. Corvallis, OR. 2012.

[18] Spool JM, Perfetti C, Brittan D. Designing for the Scent of Information: The Essentials Every Designer Needs to Know About How Users Navigate Through Large Web Sites. User Interface Engineering; 2004.

[19] Lawrance J, Bogart C, Burnett M, Bellamy R, Rector K, Fleming SD. How

programmers debug, revisited: An information foraging theory perspective. IEEE Transactions on Software Engineering. 2010 Dec 23;39(2):197-215.

[20] Lawrance J, Bellamy R, Burnett M. Scents in programs: Does information foraging theory apply to program maintenance?. In: IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007) 2007 Sep 23 (pp. 15-22). IEEE.

[21] Jin X, Niu N, Wagner M. Facilitating end-user developers by estimating time cost of foraging a webpage. In2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) 2017 Oct 11 (pp. 31-35). IEEE

[22] Srinivasa Ragavan S, Kuttal SK, Hill C, Sarma A, Piorkowski D, Burnett M. Foraging among an overabundance of similar variants. In: Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems 2016 May 7 (pp. 3509-3521).

[23] Lawrance J, Bellamy R, Burnett M, Rector K. Using information scent to model the dynamic foraging behavior of programmers in maintenance tasks. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems 2008 Apr 6 (pp. 1323-1332).

[24] Ennals R, Gay D. User-friendly functional programming for web mashups. In: Proceedings of the 12th ACM SIGPLAN international conference on Functional programming 2007 Oct 1 (pp. 223-234).

[25] Elson J, Howell J, Douceur JR. MapCruncher: integrating the world's geographic information. ACM SIGOPS Operating Systems Review. 2007 Apr 1;41(2):50-9.

[26] Wong J, Hong J. Marmite: end-user programming for the web. InCHI'06 extended abstracts on Human factors in computing systems 2006 Apr 21 (pp. 1541-1546).

[27] Automator User Guide for Mac [Internet]. Apple Support. [cited 2021Apr8]. Available from: https://support.apple.com/guide/automator/welcome/mac

[28] Faaborg A, Lieberman H. A goal-oriented web browser. In: Proceedings of the SIGCHI conference on Human Factors in computing systems 2006 Apr 22 (pp. 751-760).

[29] LaToza TD, Myers BA. Developers ask reachability questions. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1 2010 May 1 (pp. 185-194).

[30] Fitzgerald S, McCauley R, Hanks B, Murphy L, Simon B, Zander C. Debugging from the student perspective. IEEE Transactions on Education. 2009 Sep 15;53(3):390-6.

[31] Ko AJ, Myers BA. Finding causes of program output with the Java Whyline. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems 2009 Apr 4 (pp. 1569-1578).

[32] Ko AJ, Myers BA. Designing the whyline: a debugging interface for asking questions about program behavior. In: Proceedings of the SIGCHI conference on Human factors in computing systems 2004 Apr 25 (pp. 151-158).

[33] Cao J, Rector K, Park TH, Fleming SD, Burnett M, Wiedenbeck S. A debugging perspective on end-user mashup programming. In2010 IEEE Symposium on Visual Languages and Human-Centric Computing 2010 Sep 21 (pp. 149-156). IEEE.

[34] Brandt J, Guo PJ, Lewenstein J, Dontcheva M, Klemmer SR. Two studies

of opportunistic programming: interleaving web foraging, learning, and writing code. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems 2009 Apr 4 (pp. 1589-1598).

[35] Ko AJ, Myers BA, Coblenz MJ, Aung HH. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. IEEE Transactions on software engineering. 2006 Nov 30;32(12):971-87.

[36] Vessey I. Expertise in debugging computer programs: A process analysis. International Journal of Man-Machine Studies. 1985 Nov 1;23(5):459-94.

[37] Naveed MS, Sarim M. Analyzing the Effects of Error Messages Presentation on Debugging and Programming. Sukkur IBA Journal of Computing and Mathematical Sciences. 2021 Jan 5;4(2):38-48.

[38] Brandt J, Guo PJ, Lewenstein J, Dontcheva M, Klemmer SR. Two studies of opportunistic programming: interleaving web foraging, learning, and writing code. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems 2009 Apr 4 (pp. 1589-1598).

[39] Ruthruff JR, Burnett M. Six challenges in supporting end-user debugging. ACM SIGSOFT Software Engineering Notes. 2005 May 21;30(4):1-6.

[40] Phalgune A, Kissinger C, Burnett M, Cook C, Beckwith L, Ruthruff JR. Garbage in, garbage out? An empirical look at oracle mistakes by end-user programmers. In: 2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05) 2005 Sep 20 (pp. 45-52). IEEE.

[41] Kuttal SK, Sarma A, Rothermel G. On the benefits of providing versioning support for end users: an empirical study. ACM Transactions on Computer-Human Interaction (TOCHI). 2014 Feb 1;21(2):1-43.

[42] Servant F. Supporting bug investigation using history analysis. In: 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE) 2013 Nov 11 (pp. 754-757). IEEE.

[43] Myers B, Ko A. Studying development and debugging to help create a better programming environment. In: CHI 2003 Workshop on Perspectives in End User Development 2003 Apr (pp. 65-68). FL: Fort Lauderdale.

[44] Jones MC, Churchill EF. Conversations in developer communities: a preliminary analysis of the yahoo! pipes community. In: Proceedings of the fourth international conference on Communities and technologies 2009 Jun 25 (pp. 195-204).

[45] Huang AF, Huang SB, Lee EY, Yang SJ. Improving end-user programming with situational mashups in web 2.0 environment. In2008 IEEE International Symposium on Service-Oriented System Engineering 2008 Dec 18 (pp. 62-67). IEEE.

[46] Cappiello C, Matera M, Picozzi M. A UI-centric approach for the end-user development of multidevice mashups. ACM Transactions on the Web (TWEB). 2015 Jun 16;9(3):1-40.

[47] Leonard TA. Tree-sheets and structured documents (Doctoral dissertation, University of Southampton).

[48] Stolee KT, Elbaum S. Refactoring pipe-like mashups for end-user programmers. In: Proceedings of the 33rd International Conference on Software Engineering 2011 May 21 (pp. 81-90).

[49] Stolee KT, Elbaum S, Sarma A. Discovering how end-user programmers and their communities use public repositories: A study on Yahoo! Pipes. Information and Software Technology. 2013 Jul 1;55(7):1289-303.

[50] Wang HB, Wang ZH. Building Mobile News Aggregation Application with Yahoo Pipes. In: Advanced Materials Research 2013 (Vol. 756, pp. 1943-1947). Trans Tech Publications Ltd.

[51] Card SK, Pirolli P, Van Der Wege M, Morrison JB, Reeder RW, Schraedley PK, Boshart J. Information scent as a driver of web behavior graphs: Results of a protocol analysis method for web usability. In: Proceedings of the SIGCHI conference on Human factors in computing systems 2001 Mar 1 (pp. 498-505).

[52] Chi EH, Rosien A, Supattanasiri G, Williams A, Royer C, Chow C, Robles E, Dalal B, Chen J, Cousins S. The bloodhound project: automating discovery of web usability issues using the InfoScentπ simulator. In: Proceedings of the SIGCHI conference on Human factors in computing systems 2003 Apr 5 (pp. 505-512).

[53] Piorkowski D, Penney S, Henley AZ, Pistoia M, Burnett M, Tripp O, Ferrara P. Foraging goes mobile: Foraging while debugging on mobile devices. In2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) 2017 Oct 11 (pp. 9-17). IEEE.

[54] Niu N, Mahmoud A, Bradshaw G. Information foraging as a foundation for code navigation (NIER track). In: Proceedings of the 33rd International Conference on Software Engineering 2011 May 21 (pp. 816-819).

[55] Bhowmik T, Niu N, Wang W, Cheng JR, Li L, Cao X. Optimal group size for software change tasks: A social information foraging perspective. IEEE transactions on cybernetics. 2015 Apr 22;46(8):1784-95.

[56] Fu WT, Pirolli P. SNIF-ACT: A cognitive model of user navigation on the World Wide Web. Human–Computer Interaction. 2007 Nov 1;22(4):355-412.

[57] Chi EH, Pirolli P, Pitkow J. The scent of a site: A system for analyzing and predicting information scent, usage, and usability of a web site. In: Proceedings of the SIGCHI conference on Human factors in computing systems 2000 Apr 1 (pp. 161-168).

[58] Pirolli P. Information foraging theory: Adaptive interaction with information. Oxford University Press; 2007 Apr 12.

[59] Ragavan SS, Pandya B, Piorkowski D, Hill C, Kuttal SK, Sarma A, Burnett M. PFIS-V: modeling foraging behavior in the presence of variants. In: Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems 2017 May 2 (pp. 6232-6244).

[60] Lawrance J, Burnett M, Bellamy R, Bogart C, Swart C. Reactive information foraging for evolving goals. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems 2010 Apr 10 (pp. 25-34).

[61] Grigoreanu VI, Burnett MM, Robertson GG. A strategy-centric approach to the design of end-user debugging tools. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems 2010 Apr 10 (pp. 713-722).