Variation Support for End Users

Sandeep Kaur Kuttal
Computer Science and Engineering
University of Nebraska–Lincoln, Lincoln, NE, USA
skuttal@cse.unl.edu

I. Introduction

Erin is a consultant who travels frequently and wants to create a web application that may help her find hotels near her travel destination. Erin *reuses* her friend's web mashup which searches for movie theaters within a 10 mile radius of a location. She copies the application and modifies it to find the hotels in her area. While modifying, she *explores* different solutions by trying different strategies and she realizes that some of her past strategies had worked partially. She wished that she had snapshots of her past work. After a few months, on her next trip, she realizes that the current application is exhibiting unexpected behavior. The APIs used by the application have changed and the format in which the input events were originally provided is no longer supported. She needs to *debug* the application by updating her web application to reflect the new services.

Like Erin, many end-user programmers face similar problems because end-user programming environments do not facilitate users' explorations, debugging, and reuse activities. However, it is well known that end users typically learn from available examples [6] and debug their programs into existence [7].

End-user programming environments provide central repositories where users can execute and store their programs. However, these environments do not provide facilities by which users can keep track of the variations that they create for their programs. In the professional world, software developers use variation management for code reuse, program understanding, change traceability, debugging and maintenance [8]. To help users like Erin we want to bring benefits of variation management into end-user environments.

Variation management deals with variations over time and space. Variation over time involves temporal variations of a user's program. Supporting variation over time will allow developers like Erin to browse past versions of a program, determine how these versions differ from one another, and choose a particular version or revert back to a previous version. Variation over space involves (non-temporal) variations differing in program functionality at a given point in time. Supporting variation over space could help Erin in reuse as she can select an alternative variation by comparing functionality/features.

In this paper, we discuss our past research for supporting variation over time and future work for supporting variation over space.

II. Approaches

As an initial proof-of-concept for supporting variation over time we have extended Yahoo! Pipes [9]; a web mashup environment. Web mashups are situational applications that allow their users to create new web applications by combining multiple web services, data, and functionalities. Here, we summarize our past research for supporting variation over time (versioning) and debugging.

A. Variation support over time

We have supported versioning for Yahoo! Pipes, via a prototype called "Pipes Plumber". This extension keeps track of version histories of program automatically. Thus, users like Erin can utilize advantages of versioning without being aware of the underlying versioning operations such as check-in, check-out, and so forth.

Pipe Plumber enables users to interact with versions through three widgets that were added to the Yahoo! Pipes interface. These widgets are *Undo, Redo* and *History of pipes*. The first two widgets allow users to browse between consecutive versions of a pipe. The third widget, *History of pipes*, displays the modules added or removed from a pipe, per version, so that users can view the differences between versions. We performed a think-aloud study of nine participants (primarily computer science students). The study showed that the versioning support helped create pipes more effectively and efficiently [5]. The success of users in the experimental group was attributed to the fact that they were more adventurous and explored more ideas than the users in the control group, because they knew that they could go back to their previous changes.

In our next study, we wanted to verify above results with a more diverse population of users and investigate how our versioning support can aid users in performing debugging tasks. To support debugging behavior, we added another widget called *Tested* to the interface that allowed users to indicate their confidence in the pipe's correctness. *History of Pipe* list was extended by making use of colors to help users distinguish successful pipe runs, unsuccessful pipe runs, and tested versions of pipes. We hoped that this would help users more easily identify the parts of pipes that had already been tested and were therefore correct, and debug only the parts of pipes that were added later. We then conducted a controlled experiment involving 24 participants who did, and who did not, have formal programming training and experience, studying questions related to the creation and debugging of pipes [1]. Our experiment results confirmed that both groups of users could create pipes more effectively and efficiently with the aid of versioning support, and they could also debug pipes more effectively.

Next, we conducted an in-depth analysis of the data (from our previous study [1]) to statistically compare both participant

groups (i.e., computer science and end-user groups) [2]. We found that end users performed reuse tasks more effectively than the computer science group. However, the computer science group performed debugging tasks both more effectively and efficiently than the end-user group.

We further analyzed the transcripts qualitatively to investigate the motivations, thoughts, and reactions of participants toward addition of versioning support in the environment [2]. Versioning support allowed participants to be less risk-averse since they knew their changes would not be lost. Versioning also helped participants better understand the pipes and the context in which their components (modules) were used. This was primarily because it provided examples of correct usage of modules that served as a reference. Hence, participants with versioning support backtracked less than the participants without such support. Versioning also helped in debugging by reducing the testing space since participants could compare two versions and identify problem spots and focus their debugging efforts on those spots.

While versioning support for debugging helped end users "debug programs into existence" to some extent, it was not enough. End users still struggled while debugging and this necessitates additional debugging support for end-user environments. This motivated us to provide better debugging support for mashups.

B. Adding Debugging Support

Debugging is challenging in mashup environments since mashups depends on web contents, which tend to evolve. This makes mashups more vulnerable to unexpected behaviors. Moreover, the black-box nature of the programming environments adds to the complications in locating and fixing bugs since this obscures the sources of bugs, and hides distinctions between types of bugs. Additionally, the only debugging support for mashup environments currently is in the form of console output messages. Further, error messages generated are inadequate and difficult to understand by end-user programmers, which makes it difficult to fix bugs. Since reuse is a common phenomena in end-user environments, bugs can easily propagate to other mashups. To help end users like Erin debug, there is a need to support localization of bugs through automated anomaly detection and fixing of bugs by approaches that integrate various debugging strategies.

To help end users debug their created mashups we created a classification scheme of existing faults in the mashup domain, and based on this scheme we developed automated bug detection mechanisms. We then designed an interface (as an extension to Yahoo! Pipes) for supporting debugging that provided automated fault localization, used simple language for describing error messages, and cross-linked faults with error messages. We also added contextualized help and incremental assistance by providing a high-level overview of a possible solution, followed by "hints" that users can employ. This helps sustain a user's interest as they may not be overwhelmed by the number and type of errors. To investigate the use of our debugging support, we conducted an empirical think aloud study of our extension to Yahoo! Pipes with 16 web-active end users. Our study showed that it was difficult for end users to identify and localize faults without debugging support. We also observed that bugs related to program nesting, silent failures of programs, and program reuse were the hardest for end users to localize. While, debugging enhancements greatly helped mashup programmers localize and efficiently fix bugs [3].

III. Discussions and Future Work

Our studies have shown considering variation over time and automatic debugging benefits end users by enhancing their programming abilities. Based on our preliminary work, we propose several additional activities.

While considering variation over space, users will need to forage for right example to begin with. Therefore, we want to conduct user study for understanding various strategies followed by users while reusing programs from an Information foraging theory (IFT) perspective. (We have studied foraging behavior of end users while debugging from IFT perspective [4]). Some of the research questions we will like to investigate are: 1) How do users explore or search for variants? 2) How can users understand the differences among the variants that they or somebody else have created? 3) What factors affects users' decisions to select specific variants? Our overall goal of this study is to understand how end users forage for variations. We want to develop new theories and techniques to enhance the user interfaces to help them with foraging among variations.

Based on the insights gained from above user study, we will identify the features needed to create support system for variation over space. Based on these features, we will create appropriate analogous features for end-user programming environments. We will conduct a user study to investigate, to what extent does variation over space support 1) allow end users to effectively and efficiently reuse? 2) helps end users understand variations? 3) help users to create bug free programs?

Finally, we will investigate other domains to show the generalizability of our approaches beyond Yahoo! Pipes. These domains should be generalizable to most of end-user environments, which are visual in nature and support dataflow.

References

[1] S. K. Kuttal, A. Sarma, and G. Rothermel. History repeats itself more easily when you log it: Versioning for mashups. In *VLHCC*, pages 69 – 72, 2011.

[2] S. K. Kuttal, A. Sarma, and G. Rothermel. On the benefits of providing versioning support for end-users: An empirical study. In *Technical Report TR-UNL-CSE-2012-0008*. Dept. of Computer Science, U. Nebraska, 2012, available at http://cse-apps.unl.edu/facdb/publications/TR-UNL-CSE-2012-0008.pdf.

[3] S. K. Kuttal, A. Sarma, and G. Rothermel. Debugging support for end-user mashup programming. In *CHI*, pages 1609–1618, 2013.

[4] S. K. Kuttal, A. Sarma, and G. Rothermel. Predator behavior in the wild web world of bugs: An information foraging theory perspective. Preprint available on http://cse.unl.edu/~skuttal/vlhcc2013.pdf, 2013, (accepted).

[5] S. K. Kuttal, A. Sarma, A. Swearngin, and G. Rothermel. Versioning for mashups - an exploratory study. In *ISEUD*, pages 25 – 41, 2011.

[6] H. Lieberman, F. Patern, and V. Wulf. *End User Development*. Springer, first edition, 2006.

[7] M. B. Rosson and J. M. Carroll. Active programming strategies in reuse. In *ECOOP*, pages 4–20, London, UK, 1993. Springer-Verlag.

[8] W. F. Tichy. RCS-A system for version control. In *SPE*, pages 637–654, 1985.

[9] Yahoo! Pipes: http://pipes.yahoo.com/pipes/.