Creativity Support in Authoring and Backtracking

Brad A. Myers, Stephen Oney,
Human-Computer Interaction Inst.
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
{bam, soney}@cs.cmu.edu

YoungSeok Yoon,
Inst. for Software Research
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
youngseok@cs.cmu.edu

Joel Brandt
Adobe Research
San Francisco, CA 94103
joel.brandt@adobe.com

ABSTRACT
The "Natural Programming" group has been working for 15 years on making it easier for all kinds of programmers to be creative when writing software. Recently, one focus has been enabling "end-user programmers" (EUPs) such as interaction designers to more easily author interactive behaviors for the web. In a separate project, we are adding features to a code editor to support "backtracking" — undoing operations to partially or fully restore the code to a previous state — since creative exploration usually involves both moving forward to new designs and going backwards to retract some or all of the design that is not desired. In all of these projects, we seek to measure both the usability of our tools, and their effectiveness at fostering creativity.

Author Keywords
Interactive Behaviors; End-User Programming; Exploratory Programming; Backtracking; Natural Programming; Creativity.

ACM Classification Keywords
H.5.2 [Information Interfaces and Presentation]: User Interfaces – Interaction styles; H.1.2 [User/Machine Systems]: Software psychology; D.2.6 [Programming Environments]: Integrated environments

General Terms
Human Factors; Design; Measurement.

INTRODUCTION
Buxton quotes Linus Pauling as having said: "The best way to have a good idea is to have lots of ideas" [4, p. 121]. Design education places much emphasis on this strategy [6] and research suggests that exploring multiple ideas helps improve creativity [2, 8]. Moreover, creativity theory suggests the need to produce a plethora of ideas in order to arrive at the creative ones, a concept called *ideational fluency* [13]. Donald Schön, one of the most influential design theorists [11], characterizes the creative process as a conversation with materials [32]. In this conversation, designers advance the work by reflecting both in and on their actions, and by engaging with materials that specifically support conceiving and refining ideas as well as with the target material a product will be made out of. Designers reflect *in* action by evaluating and experimenting with what they are working on while they are working on it. A similar observation was made by Rosson & Carroll when studying Smalltalk programmers: a key way in which programmers create programs is to write some code, see if it does what is desired, and if not, entirely or selectively remove part of what they have created; a process they call "debugging into existence" [31]. When the process of trying out designs is embodied in writing software, it has been called *exploratory programming* [33] or *opportunistic programming* [3]. Such explorations require both forward and backward moves: forward to create new software, and backwards so the code returns at least partially to the way it was previously, either by removing inserted code or by restoring removed code. We call such backward moves *backtracking*. Besides directly helping programmers remove unwanted edits, we claim that programmers will feel more comfortable exploring if they know they have effective tools for backtracking.

The Natural Programming Project [24] tries to make programming easier by making it more *natural*, by which we mean closer to the way people think about their tasks. One way to define programming is the process of transforming a mental plan into one that is compatible with the computer [14]. The closer the programming language is to the developer's original plan, the easier this refinement process will be [12]. We have adapted a variety of HCI techniques to help understand and evaluate how developers program and use novel and conventional development tools, including Contextual Inquiry field studies [19], surveys [25], heuristic analysis and cognitive walkthrough [10], lab usability studies [9], paper prototyping [20], and *A vs. B* user studies [17].

Over the years, a number of our tools have particularly focused on the issue of *creativity*, especially as it relates to professional programmers and also "end-user programmers" (EUPs) [15], who are people who program in order to achieve some goal other than the programming itself, for example, interaction designers testing out an idea. Programmers and interaction designers often need to be creative, and it would be useful to have a way to measure the extent to which their development tools enable creativity.

To appear in:
ECSE 2013: evaluation methods for creativity support environments
Workshop at CHI'13, April 28, 2013, Paris, France.

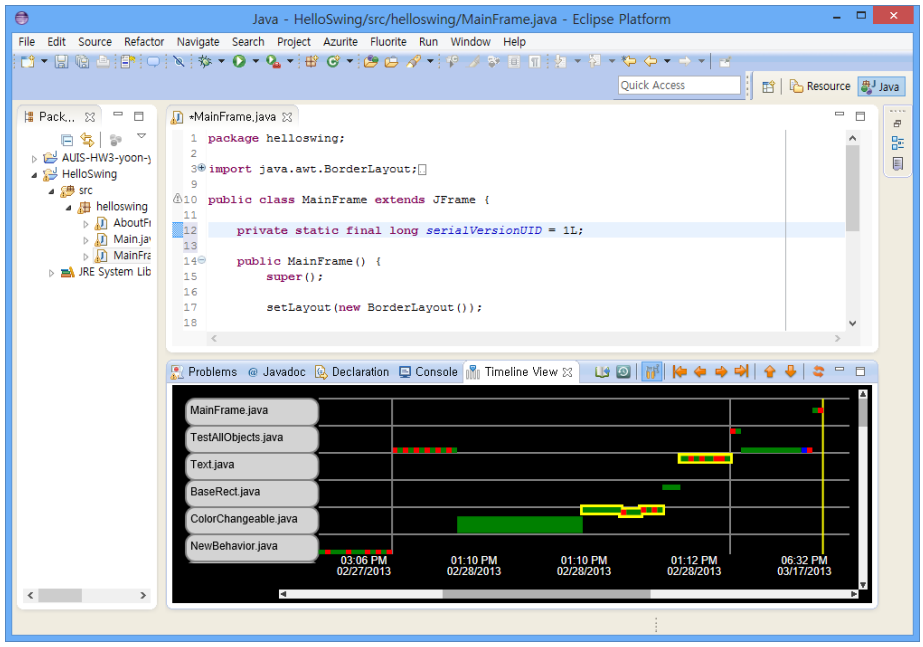

Figure 1: The current interface of AZURITE for the Eclipse IDE. The top window is the Eclipse code view, and the bottom timeline visualization shows insert (green), delete (red) and replace (blue) operations for each of the files. A vertical gray line divides the two consecutive editing sessions. The yellow vertical line at the right shows the current time. The user can select certain operations (here shown with a yellow outline), either by clicking on them, or querying in the code window for all the operations affecting a range of code, and then can selectively undo only those operations.

Current measures of creativity that we have used are all *indirect* – measuring other things that may be said to correlate with creative behavior. For example, in an early paper, we measured the number of different designs that users were able to create [18], and recently we have proposed that measuring how quickly users can move from one design to another would correlate with the success of the creativity support. However, it would certainly be useful to have a more *direct* measure of the level of creativity supported by our tools. For example, we might compare our AZURITE tool (see below) with a "preview" system in graphical editing tasks [34] or examine the tools with "sketch" like interactivity [35].

OVERVIEW OF OUR CREATIVITY SUPPORT TOOLS

Authoring

We have worked on many different kinds of authoring tools that can be considered "creativity support environments" (CSEs), at least for professional programmers and EUPs. For example, we developed many interactive tools to enable user interfaces to be created with little or no programming by user interface specialists. Some examples are Peridot [22] for creating controls (widgets like menus and scroll bars), Gamut [21] for defining behaviors by example, Silk [18] for sketching interfaces and having them automatically converted into code, HANDS [29] which is a novel programming language for kids, and Citrus [16] which is a toolkit for creating user interfaces for structured data.

Our current project for authoring is called EUCLASE, and is based on research on how interaction designers naturally express user interface behaviors, such as how the objects on the screen respond to the user [30]. We also studied how designers collaborate and express their ideas [28]. We then used these results to create a new JavaScript toolkit for creating interactive behaviors for the web, called ConstraintJS, which supports constraints combined with state diagrams [27]. Now, we are working on an interactive tool which combines a spreadsheet-like user interface with state diagrams, which we feel will enable interaction designers to more easily (and creatively) be able to author interactive behaviors themselves [26].

Backtracking

There is a large body of work and many research and commercial tools directed at making it easier for people to move forward from their ideas to designs to implementations, but there is surprisingly little support for directly helping people explore multiple variations (besides sketching on paper [4]). In particular, very little is available to help today's developers *backtrack*, even though developers report that backtracking is often required [36]. For example, modern integrated development environments (IDEs) do not utilize any of the sophisticated undo mechanisms that have been investigated through the years (e.g., [1, 5, 7, 23]), and only provide a simple linear undo model. As a result, developers cannot easily undo the changes that they made some time ago, or changes that are interleaved with edits that are still desired, but only can undo the most recent changes in the command history. Also, when the developer undoes several steps backwards and makes a new change from that point, all the previously undone commands are

discarded and cannot be redone, because the undo model only keeps a linear list instead of a command history tree. In our previous survey [36], programmers reported that they use undo mostly to remove typos or repair minor mistakes in the very last edits made. Another possible way to backtrack is to use a Version Control System (VCS), but this is not always adequate: it only works if the user thought to commit the desired version, which may not always be the case, and it is often too heavy-weight for small experiments. Furthermore, if the user has made edits that need to be retained mixed in with the edits to be backtracked, neither linear undo nor version control can be used.

We are developing a plug-in called AZURITE for the Eclipse IDE, which enables users to more easily backtrack (see Figure 1). It allows users to perform *selective undo* of only the desired operations, so users can choose exactly which operations should be undone. We address two main problems of providing selective undo in the context of text editing of code: first, we provide a way to deal with *conflicts* when a later edit overlaps an earlier one, and to effectively ask the user's intent when the conflicts cannot be resolved automatically. Second, we provide a novel way for users to *find* the operations they want to undo using a timeline visualization of the code editing history (see Figure 1). In addition, unlike other existing undo models, our selective undo model allows users to select multiple edit operations at the same time, which is very effective at minimizing the occurrence of unresolvable conflicts.

Although the timeline visualization provides a way to select multiple operations, it becomes difficult to manually select the right operations to undo as the history gets bigger. Therefore, we are working on a sophisticated *search* mechanism to allow users to easily find the operations that they want to undo using whatever they remember about those operations. We already support what we found to be the most desired operation [36]: users can search for all edits performed on a particular area of code, and undo them. We also allow users to find code which *used* to exist but has subsequently been deleted. History search not only helps the users to select the right operations to undo, but also minimizes the irresolvable conflicts because the conceptually-related edits are likely to be performed on the same area of code, and thus they usually have conflicts only among themselves. In the future, we propose to also support other interesting ways to search the history, including to find the times when a particular line, method, or class was edited; to find the last time the application compiled without error or was run without raising an exception; to find when a particular editing operation was performed (e.g., "a refactor using Extract Superclass"); etc.

Although currently implemented in the context of a code editor, AZURITE should be directly applicable for any text editor, such as Word or Pages. We feel that our backtracking ideas would also transfer to other kinds of editors as well, including design programs like Photoshop, Illustrator or even PowerPoint and Keynote, where users now need to backtrack but have little support.

FUTURE EVALUATIONS

We have been building and evaluating our research prototypes iteratively, with evaluation results driving the design of future prototypes. Currently, the main focus of the evaluations is whether the tools are *usable* by the intended audiences, and whether users are *effective* at performing the tasks we are attempting to support. These evaluations use conventional usability think-aloud and *A vs. B* lab studies. In the future, we plan to deploy our prototypes to understand how they are used in practice.

We would also like to measure the extent to which our prototypes help users be more creative. With EUCLASE, we can measure whether it helps designers create novel interactive behaviors. With AZURITE, which records everything that users do, we can have a direct measure of how often people make alternative designs, and how often they backtrack. However, both of these are indirect measures, so we are interested in using other evaluation techniques and measures as well.

ACKNOWLEDGMENTS

Funding for this research comes in part from Adobe, in part from the Korea Foundation for Advanced Studies (KFAS), and in part from NSF grant IIS-1116724. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of Adobe, KFAS or the NSF.

REFERENCES

1. Berlage, T., "A Selective Undo Mechanism for Graphical User Interfaces Based on Command Objects." *ACM Transactions on Computer Human Interaction*, 1994. **1**(3): pp. 269-294.
2. Brandt, J., *et al.*, "Example-Centric Programming: Integrating Web Search into the Development Environment," in *CHI'2010: ACM Conference on Human Factors in Computing Systems*, April, 2010. Atlanta, GA. pp. 513-522.
3. Brandt, J., *et al.*, "Opportunistic Programming: Writing Code to Prototype, Ideate, and Discover." *IEEE Software*, 2009. pp. 18-24.
4. Buxton, B., *Sketching User Experiences: Getting the Design Right and the Right Design.* 2007, San Francisco, CA: Morgan Kaufmann.
5. Cass, A.G. and Fern, C.S.T., "Modeling Dependencies for Cascading Selective Undo.," in *IFIP INTERACT 2005 Workshop on Integrating Software Engineering and Usability Engineering*, 2005.
6. Cross, N., *Design Thinking: Understanding How Designers Think and Work [Paperback]*. 2011, Berg Publishers.
7. Cubitt, T., *Undo-Tree.El Version 0.2 for Emacs.* 2010. http://www.dr-qubit.org/undo-tree/undo-tree-0.2.el.
8. Dow, S.P., *et al.*, "Parallel Prototyping Leads to Better Design Results, More Divergence, and Increased Self-

Efficacy." *ACM Trans. Comput.-Hum. Interact.*, 2010. **17**(4): pp. 1-24.
9. Ellis, B., Stylos, J., and Myers, B. "The Factory Pattern in Api Design: A Usability Evaluation," in *International Conference on Software Engineering (ICSE'2007).* 2007. Minneapolis, MN: pp. 302-312.
10. Faulring, A., *et al.*, "Agent-Assisted Task Management That Reduces Email Overload," in *International Conference on Intelligent User Interfaces: IUI'2010*, February, 2010. Hong Kong, China. pp. 61–70.
11. Forlizzi, J., Zimmerman, J., and Stolterman, E. "From Design Research to Theory: Evidence of a Maturing Field," in *International Assoc. of Societies of Design Research Conference.* 2009. Seoul, Korea: IASDR.
12. Green, T.R.G., "Cognitive Dimensions of Notations," in *People and Computers V*, A. Sutcliffe and L. Macaulay, Editors. 1989, Cambridge University Press. Cambridge.
13. Guilford, J.P., *Intelligence, Creativity and Their Educational Implications.* 1968, Robert R. Knapp.
14. Hoc, J.-M. and Nguyen-Xuan, A., "Language Semantics, Mental Models and Analogy," in *Psychology of Programming*, J.-M. Hoc, *et al.*, Editors. 1990, Academic Press. London. pp. 139-156.
15. Ko, A.J., *et al.*, "The State of the Art in End-User Software Engineering." *ACM Computing Surveys*, 2011. **43**(3): pp. Article 21, 44 pages.
16. Ko, A.J. and Myers, B.A. "Citrus: A Toolkit for Simplifying the Creation of Structured Editors for Code and Data," in *UIST'05: ACM Symposium on User Interface Software and Technology.* 2005. Seattle, WA: pp. 3-12.
17. Ko, A.J. and Myers, B.A., "Finding Causes of Program Output with the Java Whyline," in *CHI'2009: Human Factors in Computing Systems*, April 4-9, 2009. Boston, MA. pp. 1569-1578.
18. Landay, J.A., *Interactive Sketching for the Early Stages of User Interface Design.* Ph.D. Thesis, Computer Science Department, Carnegie Mellon University, 1996, CMU-HCII-96-105.
19. LaToza, T.D. and Myers, B., "Developers Ask Reachability Questions," in *ICSE'2010: Proceedings of the International Conf. on Software Engineering*, May 2-8, 2010. Capetown, South Africa. pp. 185-194.
20. LaToza, T.D. and Myers, B.A., "Visualizing Call Graphs," in *VL/HCC'2011: IEEE Symposium on Visual Languages and Human-Centric Computing*, Sept. 18-22, 2011. Pittsburgh, PA. pp. 117-124.
21. McDaniel, R.G. and Myers, B.A. "Getting More out of Programming-by-Demonstration," in *Proceedings CHI'99: Human Factors in Computing Systems.* 1999. Pittsburgh, PA: pp. 442-449.
22. Myers, B.A. "Creating Dynamic Interaction Techniques by Demonstration," in *CHI+GI'87: Human Factors in Computing Systems.* 1987. Toronto, Ont., Canada: pp. 271-278.
23. Myers, B.A. and Kosbie, D., "Reusable Hierarchical Command Objects," in *CHI'96: Human Factors in Computing Systems*, April 14-18, 1996. Vancouver, BC, Canada. pp. 260-267.
24. Myers, B.A., Pane, J.F., and Ko, A., "Natural Programming Languages and Environments." *Communications of the ACM*, 2004. **47**(9): pp. 47-52.
25. Myers, B.A., *et al.*, "How Designers Design and Program Interactive Behaviors," in *2008 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC'08*, Sept 15-18, 2008. Herrsching am Ammersee, Germany. pp. 185-188.
26. Oney, S., Myers, B., and Zimmerman, J. "Visions for Euclase: Ideas for Supporting Creativity through Better Prototyping of Behaviors," in *ACM CHI 2009 Workshop on Computational Creativity Support.* 2009. Boston, MA:
27. Oney, S., Myers, B.A., and Brandt, J., "Constraintjs: Programming Interactive Behaviors for the Web by Integrating Constraints and States," in *UIST'2012: ACM Symposium on User Interface Software and Technology*, October 7-10, 2012. Cambridge, MA. pp. 229-238.
28. Ozenc, K., *et al.*, "How to Support Designers in Getting Hold of the Immaterial Material of Software," in *CHI'2010: Human Factors in Computing Systems*, April 10-15, 2010. Atlanta, GA. pp. 2513-2522.
29. Pane, J.F., Myers, B.A., and Miller, L.B., "Using Hci Techniques to Design a More Usable Programming System," in *IEEE 2002 Symposia on Human Centric Computing Languages and Environments (HCC 2002)*, , September 3-6, 2002. Arlington, VA. pp. 198-206.
30. Park, S., Myers, B., and Ko., A. "Designers' Natural Descriptions of Interactive Behaviors," in *2008 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC'08.* 2008. Herrsching am Ammersee, Germany: pp. 185-188.
31. Rosson, M.B. and Carroll, J.M., "The Reuse of Uses in Smalltalk Programming." *ACM Transactions on Computer-Human Interaction*, 1996. **3**(3): pp. 219-253.
32. Schön, D., *The Reflective Practitioner.* 1983, London: Temple Smith.
33. Sheil, B., "Environments for Exploratory Programming." *Datamation*, 1983. reprinted in in "Papers on Interlisp-D," Sheil, B.A. and Masinter, L.M., eds., Xerox PARC Tech Report CIS-5.
34. Terry, M. and Mynatt, E.D., "Side Views: Persistent, on-Demand Previews for Open-Ended Tasks," in *UIST'2002: 15th annual ACM symposium on User interface software and technology*, 2002. ACM: Paris, France. pp. 71-80.
35. Yamamoto, Y. and Nakakoji, K., "Interaction Design of Tools for Fostering Creativity in the Early Stages of Information Design." *International Journal of Human-Computer Studies*, 2005. **63**(4-5): pp. 513-535.
36. Yoon, Y. and Myers, B.A., "An Exploratory Study of Backtracking Strategies Used by Developers," in *Cooperative and Human Aspects of Software Engineering (CHASE'2012), An ICSE 2012 Workshop.*, June 2, 2012. Zurich, Switzerland. pp. 138-144.