Rehearse: Coding Interactively while Prototyping

William Choi, Joel Brandt, Scott R. Klemmer
Stanford University HCI Group
Computer Science Department, Stanford, CA 94305, USA
{wchoi, jbrandt, srk}@cs.stanford.edu

ABSTRACT
To encourage design thinking while prototyping in code, development tools should help programmers in both "getting the right code" and "getting the code right." To support these principles, we introduce *Rehearse*, a source code editor that enables *interactive development* of JavaScript with immediate evaluation and infinite undo of execution. The system demonstrates a tight coupling of two traditionally separate tools: a source code editor and a read-eval-print loop. A first-use study confirmed the usefulness of the tool in alleviating frequent edit-debug cycles inherent in prototyping, but such a tool must be integrated into existing practices in order to be adopted.

ACM Classification: H.5.2 [Information interfaces and presentation]: User Interfaces – Graphical user interfaces; D.2.6 [Programming environments]: Interactive environments

General terms: Design, Human Factors

Keywords: Prototyping, interactive development, undo, design thinking, design tools

INTRODUCTION
Prototyping is a design thinking task characterized by exploration of alternatives and iterative cycles of development and evaluation [3]. However, prototyping *in code* is often a challenge because current tools do not foreground these ideals. For example, when creating functional prototypes, programmers often rapidly switch between writing a few lines of code and testing those lines – in a recent study, we found that during prototyping 50% of all edit-debug cycles were less than 30 seconds in length [1]. In these cycles, a great deal of time is spent getting the program to a state where the new lines can be tested. We suggest that this evaluation burden hinders the prototyping process: programmers *settle on code that works* but does not fully explore their current idea, and *settle on an idea that is implemented* when exploring more ideas would be beneficial.

This disconnect motivates our ongoing work to understand how tools can better encourage design thinking while prototyping in code. We hypothesize that design thinking during programming comprises two related tasks: getting the code right (I know what I want to implement; does the code I wrote implement that?), and getting the right code (was what I implemented the right thing to implement?).

UIST'08, October 19-22, 2008, Monterey, California, USA
ACM 978-1-59593-975-3/08/10

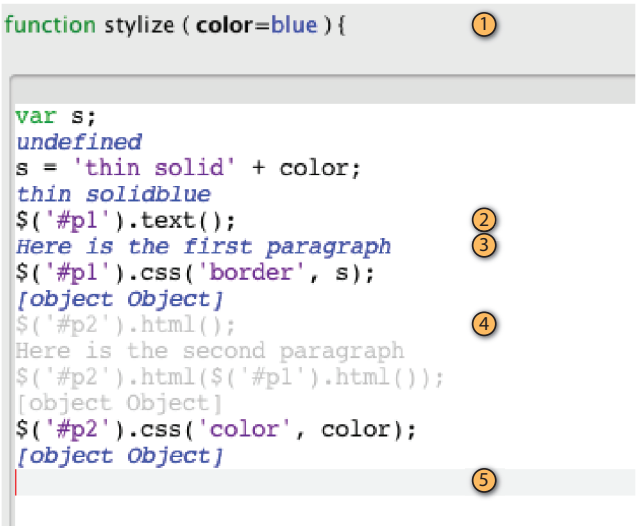

Figure 1: A screenshot of the *Rehearse* editor. (1) The function declaration, parameter names, and current values; (2) a statement that has been executed and (3) the result of execution; (4) an undone statement; (5) the current line.

Getting the code right: Tools need to decrease the amount of time between authoring a statement and evaluating the effect of that statement.

Getting the right code: Tools need to make it easier to backtrack and explore multiple paths. Tools should minimize the cost of deciding to put the current approach on hold and try a new one.

We are building *Rehearse*, a tool that explores new user interfaces for programming with these ideas in mind. We describe the first version of this tool, present the results of a first-use study, and discuss future research directions.

REHEARSE
Rehearse is a tool that supports *interactive development* in JavaScript. Using the tool, programmers can specify that a certain piece of code—a JavaScript function in this case—will be defined *interactively*, deferring the actual writing of the code until the first time it would be executed.

When the function is called for the first time, the user is automatically presented with the interactive editor (Figure 1), which shows the pre-defined function and parameter names as well as actual values of the parameters passed in.

The interactive editor resembles a read-eval-print loop in that each statement is immediately executed within the cur-

rent scope and the effects of that statement are immediately visible in the execution environment (here, the web browser). The string representation of the result of the statement is also displayed in the editor. This immediate execution and feedback helps programmers with the task of "getting the code right."

Rehearse goes beyond traditional read-eval-print loops in two ways. First, it allows programmers to undo executed lines, leading to an easier exploration of multiple alternatives and thus assisting the task of "getting the right code." Second, the interactive editor is tightly integrated with the standard development process: when the user is done writing a function interactively, *Rehearse* places the function definition in the appropriate source code file and injects the function into the browser's DOM so that development can proceed uninterrupted. This gives the programmer the flexibility of implementing some functionality interactively and other functionality in the traditional manner.

RELATED WORK

Many self-contained systems for teaching computer science, such as Alice [2], employ immediate evaluation to make development a more interactive experience. While these systems demonstrate the effectiveness of interactive development, it is typically difficult to build large applications inside these systems because *all* code must be written interactively. Similarly, many languages, such as Python and LISP, provide interactive read-eval-print loops to aid experimentation and testing. These environments, however, are transient, making it difficult to integrate what has been created into a larger project. *Rehearse* extends these two ideas, allowing users to implement *some* functionality interactively, and then feed this functionality back into the system they are building.

Rehearse also builds on many ideas from testing and debugging tools. For example, the Omniscient Debugger allows users to step backward in time after a breakpoint is reached [4]. *Rehearse* transfers this idea from code debugging to code authoring, allowing users to undo and redo execution during development. Rothermel et al.'s work on "What You See is What You Test" (WYSIWYT) explores visual representations to help guide users when debugging spreadsheets [5]. The visual feedback provided by *Rehearse* is similar at a high level: *Rehearse* offers both *spatial* (the result of a statement is displayed below the statement) and *temporal* (the statement is executed and its effects are seen immediately) feedback for every statement written to help guide debugging.

USER STUDY

We ran a small first-use study to help guide our future work on *Rehearse*. Four experienced web programmers participated in a one-hour session. Participants were Masters and Ph.D. students in Computer Science, and all had at least 6 years of programming experience. Participants were asked to add two features to a simple web-based forum: a *reply to this post* feature where the reply user interface was added dynamically and the reply was submitted using AJAX, and a *display profile* feature that would display a pop-up window showing a user's profile retrieved via AJAX.

The first task was completed with guidance from one of the researchers (while another observed) in order to orient the participant to the system; the second task was completed by the participant on her or his own. Participants were asked to think aloud during their work. When participants expressed confusion or frustration, we broke into a brief participatory design session to more deeply understand the breakdown.

FINDINGS & FUTURE WORK

Based on qualitative data gathered during the first-use study, we have identified three directions for future work:

1. Interactions in *Rehearse* must match existing programming practices: Much of the confusion users experienced stemmed from the differences between *Rehearse* and traditional editors. Common operations, like inserting or reordering a line in the middle of a code block, were difficult to do in inside *Rehearse*: users were required to undo to the point at which they wanted to make the edit, and then redo subsequent statements. We are currently extending *Rehearse* to allow users to edit any line by performing the necessary undo and redo operations in the background.

2. Users must be able to fluidly move between *Rehearse* and traditional tools: In the current system, users cannot make changes with their traditional editor while defining a function interactively. In practice, users often want to edit multiple functions at the same time (*e.g.*, to modify a function they are about to call). This suggests integrating *Rehearse* into the traditional editor: statements written in the currently executing function should be executed interactively, and statements written elsewhere should be injected into the system, but not executed until the appropriate time.

3. While *Rehearse* supports the exploration of multiple paths at the statement level, support for exploration at the "feature" level is also needed: *Rehearse* allowed programmers to explore multiple solutions at the several-statement level through undo and redo. However, there is currently no support for "undo" or "redo" of larger changes. This makes it difficult to, for example, experiment with several wholly different interactions. We are currently working on an extremely lightweight version control tool to complement *Rehearse* that will address this need.

REFERENCES

1 Brandt, J., P. Guo, J. Lewenstein, and S. Klemmer. Opportunistic Programming: How Rapid Ideation and Prototyping Occur in Practice. In *WEUSE: Workshop on End-User Software Engineering*, Leipzig, Germany, 2008.

2 Conway, M., et al. Alice: lessons learned from building a 3D system for novices. In Proceedings of *CHI: ACM Conference on Human Factors in Computing Systems*, The Hague, The Netherlands, 2000.

3 Hartmann, B., S. Klemmer, M. Bernstein, et al. Reflective Physical Prototyping through Integrated Design, Test, and Analysis. In Proceedings of *UIST: ACM Symposium on User Interface Software and Technology*, Montreux, Switzerland, 2006.

4 Lewis, B. Omniscient Debugger. http://www.lambdacs.com/debugger/, 2003.

5 Rothermel, K., et al. WYSIWYT Testing in the Spreadsheet Paradigm: An Empirical Evaluation. In Proceedings of *ICSE: Inernational Conference on Software Engineering*, Limerick, Ireland, 2000