A Contextual Inquiry of Expert Programmers in an Event-Based Programming Environment

Amy J. Ko
Human-Computer Interaction Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213 USA
ajko@cmu.edu

ABSTRACT
Event-based programming has been studied little, yet recent work suggests that language paradigm can predict programming strategies and performance. A contextual inquiry of four expert programmers using the Alice 3D programming environment was performed in order to discover how event-based programming strategies might be supported in programming environments. Various programming, testing, and debugging breakdowns were extracted from observations and possible programming environment tools are suggested as aids to avoid these breakdowns. Future analyses and studies are described.

Keywords
Programming environment, contextual inquiry, event-based

INTRODUCTION
Event-based programming, in which events drive the majority of a program's runtime behavior, is increasingly common in modern programming languages. Visual Basic, Java, Macromedia's Director, as well as many recent research prototype languages such as Alice [1] provide event-based constructs and user interfaces, enabling programmers to efficiently create highly interactive environments. Yet very few studies of programming environments and language usability [4] investigate event-based programming languages. Since recent studies suggest that language paradigm is a predictor of program programming strategies and performance [2][5], environments for event-based programming languages may need special support for event-based programming.

This study investigates expert programmers using Alice with the method of contextual inquiry (CI) [3] in order to identify problems that programmers encounter when creating interactive, event-based programs and to assess the utility of CIs for extracting design requirements for programming environments. From these observations, we describe tools that could help programmers create, test, and debug event-based programs more effectively.

CHI 2003, April 5-10, 2003, Ft. Lauderdale, Florida, USA.
ACM 1-58113-637-4/03/0004

METHOD
Participating programmers were enrolled in the "Building Virtual Worlds" course offered at Carnegie Mellon University. The course requires collaborations among programmers, modelers, sound engineers, and painters to create a new interactive 3D world every two weeks using Alice (see Figure 1). Alice provides a limited object model, global event handlers, and a strictly enforced structured editor, preventing all syntax errors.

Four expert programmers were recruited and observed during the second half of the semester, after the programmers were experienced with Alice. Other than Alice, the least expert programmer had experience with 3 languages and 4 environments, and the most expert had experience with 6 languages and 8 environments. The experimenter met with each programmer and explained the focus of the CI—to identify programming, testing, and debugging difficulties Alice programmers experience. As programmers worked on their programs, the experimenter recorded observations on paper and video. The experimenter formed hypotheses about the programmer's actions *in situ* and asked the programmers if the hypotheses were correct. For example, the experimenter would say, "It looks like you're trying to align these two objects." and the programmer would reply, "Basically. I want them to be

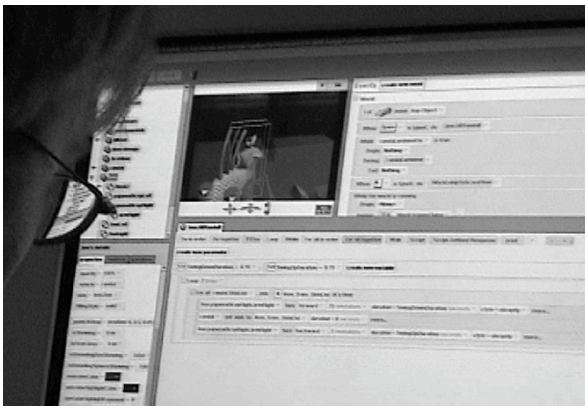

Figure 1. An Alice user participating in a contextual inquiry. This is a typical view of Alice, with code and events at the lower and upper right, the worldview at the top center, the object list at the top left and the selected object's properties at the bottom left.

Goal	Fix a hand-copied animation.
Context	A hand-copied animation was working, but the rabbit's head was moving off the body.
Strategy	Moves the camera close to the rabbit, runs the world, views the animation at varying speeds, then inspects code.
Outcome	While changing each parameter value of the copied code, he forgot to change the parameter for the head animation. Also discovered that he accidentally changed the distance parameter instead of duration.

Figure 2. An example of a breakdown scenario. A copied animation was not behaving as expected because a parameter was not changed properly.

aligned on this axis, but I don't care about the other two." Participants were paid $10 per hour for their participation.

RESULTS AND DISCUSSION

Approximately 12 hours of observations were obtained from the four programmers over 12 sessions. Each of the sessions was reviewed for *breakdown scenarios* (see Figure 2 for an example), in which a programmer's strategy was difficult to perform or unsuccessful. Breakdowns scenarios were consolidated into the problem types described below.

Programming Problems

In many breakdowns, code was reused to perform similar operations such as animations or calculations, but the code was not properly adapted for its new location (as seen in Figure 2). These bugs were particularly difficult to isolate because they propagated through complex animations, which depended on events. These breakdowns highlight the need for a *smart copy and paste* mechanism that could automatically coerce parameters from method to method.

In other scenarios, programmers needed to create finely tuned sequences of animations and events by tweaking existing code. However, programmers often reverted to a previous version of code manually, to avoid undoing intermediary changes to unrelated code. One way to alleviate this difficulty would be a *multi-level intelligent undo*, which could keep an extensive modification history for each object's methods and global event handlers.

Testing Problems

The programmers used visual cues extensively in order to aid testing tasks. For example, one programmer assigned the color of an object to the triggering of an event handler in order to verify the event occurred at the proper time. This suggests a need for a way of saying "*watch this variable by mapping it's value to* ***this***" where *this* could be a visual cue such as an object's color, size, or visibility.

A difficulty testing code in isolation was the most prevalent breakdown. As one example, programmers were forced to wait for long animation sequences to complete in order to test the end of the sequence. To tweak an animation, programmers made a small modification, wrote an event to run the animation when a key was pressed during runtime, ran the world, viewed the animation, and repeated. Also, to test a program's response to an event in a specific world state, the programmer had to manually recreate the world state, and cause the event to occur. One possible solution to these problems would be a *timeline visualization* of events and methods in the world with the ability to click and zoom on objects, events, and time periods. Programmers could then recreate problems and directly associate a world state with specific code.

Debugging Problems

Debugging breakdowns occurred when programmers had difficulty answering debugging questions of the form "when," "why," and "why *not*." Questions such as "when was the last time this object moved?" were difficult to answer; the *timeline visualization* discussed earlier could provide immediate access to this information. Answering other questions of the form "why" and "why not," were highly involved debugging tasks. The system could use simple heuristics to answer these. For example, in reply to "why did this object move?" the system could show the code that last moved the object; in response to "why did this disappear?" the system could highlight common causes in the code (resized to zero, moved out of view, etc.).

FUTURE WORK

Future work will involve further analyses of the data obtained in these CI, as well as CIs with novice and expert programmers new to Alice. Using these observations as user interface design guidelines, the tools proposed here and those that arise from future observations will be built into Alice and tested empirically. The expected results from this work are new techniques and user interfaces applicable to all programming environments supporting event-based languages.

ACKNOWLEDGMENTS

I thank Brad Myers for his guidance and the Building Virtual World students for their permission to observe.

REFERENCES

1. Conway, M. et al. Alice: Lessons Learned From Building a 3D System For Novices. *Proceedings on CHI: Human Factors in Computing Systems*, The Hague, The Netherlands, Apr 1-6, 2000, 486-493.
2. Corritore, C. and Wiedenbeck, S. An Exploratory Study of Program Comprehension Strategies of Procedural and Object-Oriented Programmers, *Intl. J. of Human-Computer Studies, 2001, 54*, 1-23.
3. Holtzblatt K. and Beyer, H. *Contextual Design: Defining Customer-Centered Systems*. Morgan Kaufmann, San Francisco, CA, 1998.
4. Pane, J. and Myers, J. Usability Issues in the Design of Novice Programming Systems, Carnegie Mellon University, School of Computer Science Technical Report CMU-CS-96-132, 1996, Pittsburgh, PA.
5. Navarro-Prieto, R. and Canas, J. Are Visual Programming Languages Better? The Role of Imagery in Program Comprehension. *Intl. J. of Human-Computer Studies, 2001,* 54, 799-829.