A Creative Programming Environment, Remixed

Henry Lieberman

MIT Media Lab, Cambridge, MA, USA

Introduction

Flashback, 1978. I'm a young researcher at the MIT Artificial Intelligence Lab. For the preceding several years, I had worked with Seymour Papert's Logo group, trying to make computer programming accessible to young children as a way of teaching them to think, and give them an environment in which they could explore, hypothesize, experiment, and understand what math and science were really about by experiencing the process themselves. What we were trying to do, essentially, was make a *Creative Programming Environment*. Remember those words.

I had become somewhat frustrated with Logo and wanted to push programming for beginners in a new direction. While graphics, using the famous Logo turtle, was a linchpin of our strategy for getting kids engaged with learning programming, Logo itself was still a textual programming language, and I wondered about the possibility of using graphics itself directly for programming. I discussed my interest with a researcher visiting the lab, and he handed me a copy of a 1975 Stanford thesis. "You ought to read this".

Ought to, indeed. The title: "Pygmalion: A Creative Programming Environment". A *Creative Programming Environment*, huh, OK, sounds up my alley. The author: one David Canfield Smith. Never heard of him. But hey, his advisor was Alan Kay. Good sign. Kay, like Papert, was one of the greats in promoting programming as a medium for children to learn and express themselves.

It was a rather strange document. It was supposed to be about a programming environment, but he didn't really even start to talk about the programming environment until page 67 of the thesis. The writing before that was filled with sections entitled things like "The Nature of Creativity", and "The Computer as an Artistic Resource". Was this a thesis in computer science or in philosophy? As it turns out, both.

Much later, Smith recounted to me his first meeting with his advisor Alan Kay. Kay handed him a big stack of books. “Oh, great”, Smith thought, "a bunch

of books on operating systems and programming methodology." Instead, it was books like Gombrich's *Art and Illusion*, Arnheim's *Visual Thinking*, and Koestler's *The Act of Creation.* Smith credits Kay for inspiring his own approach to creativity.

It took me a while to get into reading the thesis, but once I did, I tore through it in a single sitting, and emerged dazzled and stunned. This is fantastic! I went around in a daze, blabbing about it to every person I met. "Well, what's it about?" someone would naïvely ask. I didn't have a good answer. "It's about…. well, a new programming language, but um… not really a language, but a new way of doing programming… but it's really about the creative process…. um… well…. you gotta read it….".

The next year, I found myself making my first trip to Silicon Valley. I vowed to look up the author of the document. Arriving at the famous Xerox Parc, I asked a secretary for him. She said no-one of that name worked there. Are you sure? After asking around, she said, "Oh yeah, he works across the street, in, um, kind of, … another part of Xerox". I went to an unmarked building, where I was led to his office. The low-key approach, I now know, was because Smith was by then working at a very secretive Xerox division charged with developing the hush-hush Xerox Star project. Star eventually became the first modern iconic file system, which Steve Jobs then famously "borrowed" for the Macintosh. It is not an exaggeration to say that we owe this man all of today's modern graphical

interfaces. If you don't believe me on this last point, see his article (Smith et. al., 1982), which shows how the programming icons of Pygmalion could be transformed into metaphors for office objects.

Smith was welcoming and modest, seemingly flattered that someone would take the interest to look him up because of his thesis. I gathered it had not received a great deal of attention after it was completed. He had not published about it in any major conference or journal. The thesis itself was published only as a book by an obscure publisher in Switzerland (Smith, 1977).

"Would it be possible for me to actually see Pygmalion running?" I asked. He said he would try. He hadn't touched it in a long while, and he needed to revert the microcode of the Alto machine on which it ran, to an older version compatible with the program. He reached up to the top shelf, dusted off an old disk-pack, and fired it up. It ran, though a little slowly, and he demonstrated how to do Factorial of 3, that E. coli of programming language demonstrations. Then he said, "and now we'll try Factorial of 6". And it crashed. The disk had a head crash. There were no backups. I believe I am the last person (other than Smith himself) ever to actually see the program live.

But I got the idea. And it transformed me.

Pygmalion's innovations

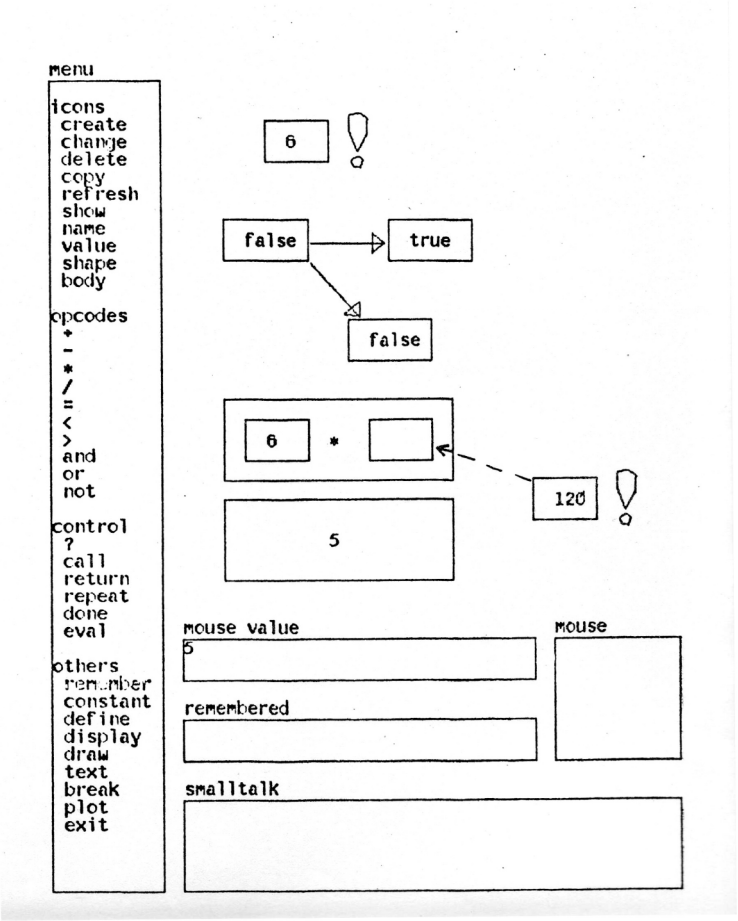

Figure 1. Pygmalion, computing Factorial of 6, from (Smith, 1975)

Pygmalion innovated in so many ways, it's not funny.

It is generally credited as the first system to introduce the modern notion of icons. At that time, the author even felt the need to explain what an icon was, "Communication between human being and computer is by means of visual

entities called 'icons', subsuming the notions of variable, reference, data structure, function, and picture".

It was certainly the first true iconic programming language. Every other attempt at graphical programming languages until then had been essentially the same as a flowchart, and even today the notion of graphical programming language is too often just what I call "icons on strings". Here icons were used not only to represent switches that invoked functions, but containment of icons represented containment in the programming language. It was probably the first system to introduce drag and drop as a way of passing arguments to functions. The thesis quite explicitly talks about the semiotics of the role of icons as symbolic representations of concepts, and the differences between textual and graphic representations. The eye-opening discussion of these roles still applies to all graphical interfaces today.

It conceived of programming as a process of animation, using successive states of the graphic display to represent successive states of the program. It understood the differences between static and dynamic representations, again a valuable principle applicable to all of today's interfaces. You could both write your program and test it at the same time.

Most importantly, it introduced the idea of *Programming by Example*, a revolutionary idea that I believe is still underappreciated today. It identified one of the major obstacles to making computers easy to use is excessive abstraction.

Programming involves languages that talk about abstract concepts, but it is hard for most people to visualize how these abstract concepts relate to the concrete behavior of the computer in an actual example; or how to achieve some desired behavior in concrete examples by specifying abstract concepts. Programming by Example is the idea that the user can just show the computer an example of what they would like to do, demonstrate the steps of the procedure on the concrete example, and the computer can record the steps, and *generalize* them to a program that can work in analogous situations.

Inspired by Pygmalion, Programming by Example has since become one of my major research topics. I went on to develop PBE systems like Tinker, Mondian (Cypher, 1993), Grammex (Lieberman, 2001), and Creo/Miro (Faaborg 2006). I was a co-editor of the first book on the topic, Allen Cypher's *Watch What I Do* (Cypher, 1993), and in 2001, I edited the book, *Your Wish is My Command* (Lieberman, 2001). I thus owe a significant part of my career to the inspiration provided by Smith's work.

But what I liked best about the thesis was that each detail presented about the system was directly motivated by the desire to support creativity: Visual programming because the creative process is intensive in diagramming and visualization; Programming by Example because the creative process works by making analogies that allow old ideas to be employed as metaphors for new

situations; interactive techniques like Drag & Drop and immediate execution, because the creative process depends on quick tinkering for experimentation.

Lessons from Pygmalion for today's HCI

What lessons can we learn from Pygmalion for contemporary research in Human-Computer Interaction? Above and beyond the specific innovations of the thesis, some of which have been thoroughly absorbed by the community (icons), others of which have yet to be fully appreciated (Programming by Example), I think Pygmalion can teach us, by example, how to have the courage to do HCI research that is truly innovative.

1. Don't let the big questions scare you. Let's go back to the title: *A Creative Programming Environment.* What chutzpah to think that your thesis could make a significant dent in the age-old problem of creativity! Yet there is a real problem there. The creative potential of computers should be obvious, yet there was (and is) no easy-enough way for a person to tap into it. Why the hell not? Get indignant about the field's failures. Fix 'em.

2. Think Globally, Act Locally. Yes, the thesis attacked big and important questions, but it didn't stay completely at the philosophical level, otherwise it really would have been a philosophy, not a science, thesis. Even though you can't entirely solve the problem of creativity in general, think about

how creativity works in the domains you're interested in. Build a system. Show people what things would be like if your vision were realized. Try it out. Reflect on the experience. Tell others what they should learn from seeing it. At my lab, now, the MIT Media Lab, our motto is: Demo or Die.

3. Hunt for the good stuff, even in out-of-the-way places. The reason I tell you the story about how I discovered Smith's work is to say that sometimes the most innovative work might be ignored by the mainstream. You might find it in long-forgotten theses instead of the Best Paper at CHI.

4. Teach by Example. Finally, let the work speak for itself in making the point you're trying to promote. The work on Pygmalion was a fantastic example of creativity in its own right. It amply supported its own ideas with rich visualizations; deep thinking about what it takes to support the creative process, and made a firm case for programming as a medium for creativity.

Conclusion

Could an article on Pygmalion get accepted to the CHI or Interact conferences today? Sadly, probably not. The reviewers would find a million things wrong with it. No user-centered requirements gathering or prototyping prior to implementation. No user testing after implementation. Only works on small examples. Won't scale up. You can imagine the criticism.

Too much of today's HCI is incremental, another-brick-in-the-wall work. Well-designed experiments, diligent implementation and testing, but ultimately, work that won't change things very much. But there should always be a place in HCI for the revolutionary, almost crackpot, work that at least holds the possibility of profoundly transforming the field. This is the story of one such work. Now go do some more.

References

Cypher, Allen, et. Al., *Watch What I Do: Programming by Demonstration*, MIT Press, Cambridge, MA, 1994.

Faaborg, Alex and Lieberman, Henry, A Goal-Oriented Browser, ACM Conference on Computer-Human Interaction (CHI-06), Montréal, April 2006.

Lieberman, Henry, *Your Wish is my Command: Programming by Example*, Morgan Kaufmann, San Francisco, 2001.

Smith, David Canfield, *Pygmalion, A Creative Programming Environment*, Birkhauser-Verlag, Basel, 1977. Also Stanford University AI Memo 260, CS Dept. Stan-CS-75-499, June 1975.

Smith, David Canfield, Irby, C., Kimball, R., Verplank, W., and Harslem, E. Designing the Star user interface. Byte 7,4 (Apr, 1982), 242-282