End-User Software Engineering Position Paper

Henry Lieberman
MIT Media Laboratory
20 Ames St. 384A
Cambridge, MA 02139 USA
lieber@media.mit.edu

PERSONAL WORK

My goal is to make the process of programming easier, especially for people who are not necessarily specialists in computer science. Why is it so much harder to program a computer than simply to use a computer application? I can't think of any good reason why this is so; we just happen to have a tradition of arcane programming languages and mystically complex software development techniques. We can do much better.

My background is in Human-Computer Interface and Artificial Intelligence, and my methodology is to use ideas from these fields to improve the situation. HCI has amassed an enormous body of knowledge about what makes interfaces easy to use, and this has been applied widely to many computer applications for end users. Oddly, little of this has been applied to making interfaces for programming easier to use. Non-experts tend to believe that programmers practice a kind of voodoo, perceived to be naturally arcane and mysterious. Since they can handle it so well, programmers aren't perceived as needing ease of use. But we all pay the price for this misconception.

Programming is the art of teaching new behavior to a computer. It's really the same problem as machine learning, which is where AI comes in. I believe the route to making programming easier is to make the computer smarter, make it capable of learning, and capable of accepting direction in the way that users feel most comfortable expressing it.

To that end, I've been exploring the following topics, among others:

Programming in Natural Language

Programming languages are a stumbling block for most beginning programmers. Why not just express what you want in English? Many believe this goal to be infeasible, but natural language understanding has made vast progress in recent years. We can use partial understanding, mixed initiative dialog models, and Commonsense reasoning to, at least partially, express procedural ideas in natural language [5]. I've also explored several ideas in Visual Programming, since some things are best expressed in pictures rather than words.

Programming by Example

People learn and teach best by example. But conventional programming languages require you to express procedures in the abstract, rather than through examples. I'd like to see the ability to demonstrate examples in concrete situations, have the system record them, and generalize them to yield a procedure capable of working on analogous examples. I've made several systems in this area, and edited a book [2].

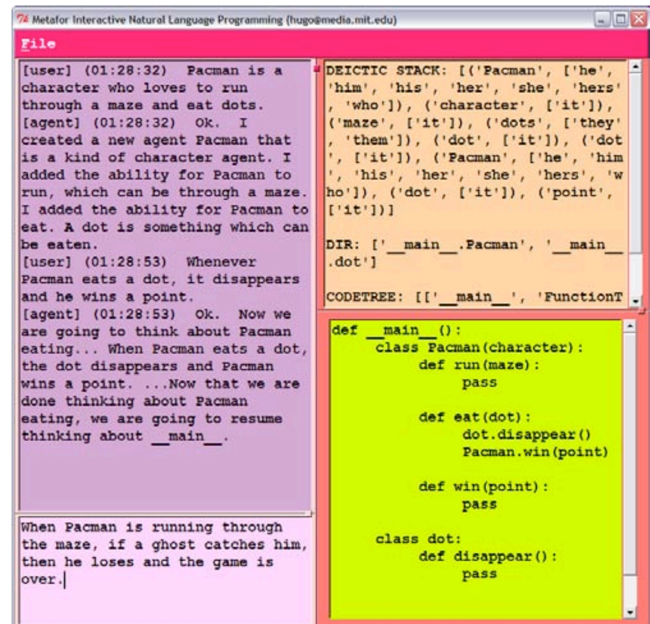

Figure 1: Metafor Natural Language Programming system

Debugging

I think the most pressing need in software development is not programming per se, but *debugging*. Programmers spend roughly half their time debugging, but debugging tools have hardly improved since the earliest days of computing. I've worked on several innovative reversible, graphical debuggers, based on ideas from diagnostic reasoning in AI. [1]

I've been exploring the idea of *end-user debugging* [3], what one might call "debugging without programming". The idea is that even when ordinary application use fails to meet the expectations of users, they could fruitfully use debugging techniques to discover what went wrong.

Dagstuhl Seminar Proceedings 07081
End-User Software Engineering
http://drops.dagstuhl.de/opus/volltexte/2007/1092

END-USER DEVELOPMENT AND SOFTWARE ENGINEERING

My book [4], edited with Fabio Paterno and Volker Wulf, I think, provides a comprehensive, and up-to-date survey of the state of the art and future directions in this area. One of the big successes of the book, I believe, was to bring together the community of people who are working to make programming easier for beginners, such as children, with people from Software Engineering who are trying to move some of their techniques into the realm of less expert users. This workshop, of course, continues that interdisciplinary effort.

For the book, we chose the title *End-User Development*, rather than "Software Engineering", though the concerns were pretty much the same as this workshop. Software Engineering has a lot to contribute in terms of bringing design methodology, collaborative programming, testing, maintenance, and other larger concerns to programming for less expert users. But I think we still have a problem positioning this effort with respect to conventional Software Engineering. For the book, I was worried that a too-close association would scare off beginning and casual users.

I commend the bravery of the workshop organizers for reaching out to the SE community. Traditional Software Engineering is really largely about the management and organization of large software projects in industry. Sometimes they are not so sympathetic to efforts to make programming easier, because you don't want to make it too easy to modify a large software project. The risk of errors or miscoordination is too great. Reliability and efficiency sometimes trump ease of development, for applications like banks and airline reservations. The approaches advocated by SE are much too heavyweight for beginning users – bureaucratic design methodologies, abstract formal verification – at least, without some radical rethinking.

I think the ideal to shoot for, first, is to maximize ease of getting started in the beginning. Program development should be as informal, flexible, lightweight, agile, and dynamic as we can possibly make it. Formal methodologies be damned. If we can do this, we can make programming accessible to millions of people who are now scared to death of it.

But as programs (or their developers) mature, some of the legitimate concerns of Software Engineering do indeed come into play, even for nonprofessional users. Programs may then need to be maintained, tested, extended, shared with others, etc., in which case Software Engineering techniques could potentially yield benefits. But we shouldn't just, as conventional SE does, wag our fingers at the users, "Ya shoulda done it right in the first place".

I think our challenge is to figure out how to *smoothly* go from the initial conception of a project, vague and imprecise at it must necessarily be, to only gradually introduce more structured representations and abstract tools, all the while without placing undue burdens on the user. To do this, I think we have to give up the idea of a single representation for programs, be it a programming language or something else.

I also think that this process of solidifying a program should be *reversible,* so that any point, one can return to the more informal forms without needless loss of effort. This will encourage the user to learn new insights from the process of software development without feeling like they get stuck by their sunk investment in an initial approach.

Finally, as much as is possible, we should make this process as *automatic* as we can, though the use of program transformation, dependency maintenance, automated reasoning, mixed-initiative interfaces, visualization, and machine learning. Otherwise, I think it will be too much overhead for a non-expert user themselves to keep track of the myriad facets that software development entails. If we succeed in this, people will become End-User Software Engineers without their even realizing it.

REFERENCES

Most of these references can be found at:

http://www.media.mit.edu/~lieber/Publications/Publications.html

1. H. Lieberman, ed. Special Issue on *The Debugging Scandal*, Communications of the ACM, April 1997.
2. H. Lieberman, ed., Your Wish is My Command: Programming by Example, Morgan Kaufmann, 2001.
3. E. Wagner and H. Lieberman, End User Debugging for Electronic Commerce, ACM Conference on Intelligent User Interfaces, Miami Beach, January 2003.
4. H. Lieberman, F. Paterno and V. Wulf, eds. *End-User Development*, Springer Academic Publishers, 2006.
5. H. Lieberman and H. Liu, Metafor: Visualizing Stories as Code, ACM Conference on Intelligent User Interfaces (IUI-2005), San Diego, January 2005