

End User Programming / Informal Programming

(Original title: End-User Programming and Blended-User Programming)

Howie Goodell, Sarah Kuhn, David Maulsby, and Carol Traynor

ABSTRACT

The phenomenon we call "programming" is not limited to regular professional programmers. Farthest away are end user programmers, who program just as a means to an end in their own professions. In between are people with very diverse backgrounds, general education, and programming knowledge. Like professional programmers, they write programs for others as their major job activity. Yet they also share characteristics of end user programmers, such as a strong specialization in one application area. All these "informal programmers" need more study and support to help them maximize their contributions.

Keywords

End user programming (EUP), end user, informal programming, Programming By Example (PBE), Programming By Demonstration (PBD), domain-specific programming languages, Artificial Intelligence (AI), agents, natural programming, software user communities, teaching programming, learning programming, open source, XML, Participatory Design (PD).

INTRODUCTION

Fifteen people attended the workshop: two organizers, ten workshop participants, and three invited discussants. Workshop participants had each submitted a position paper prior to being accepted into the workshop.

MORNING SESSION 1a: POSITION PAPERS

Each participant had 3 minutes to introduce themselves, state what they hoped to gain from the workshop, and sum-

marize their position paper. The full text of these papers is available online [1].

1. "Intelligent Support for Testing in Languages for Informal Programmers", presented by Margaret Burnett (Oregon State University).
2. "Practical End User Programming with Stagecast Creator", presented by Allen Cypher (Stagecast Software Inc.: Palo Alto CA).
3. "Mapping the Terrain of End User Programming", presented by Sarah Kuhn (University of Massachusetts Lowell: Lowell MA).
4. "A practical and empirical approach for biologists who almost program", presented by Catherine Letondal (Pasteur Institute: France).
5. "Interfaces that Give and Take Advice", presented by Henry Lieberman (MIT: Cambridge MA).
6. "Do End Users Program?" presented by David Maulsby (24C Group Inc.: Calgary Canada).
7. "Studying End User Programming", presented by Brad A. Myers and John F. Pane (Carnegie Mellon University: Pittsburgh PA).
8. "Supporting End-User Programming by Teachers in the Public Schools", presented by Mary Beth Rosson (Virginia Tech).
9. "Settling for Less than the Holy Grail", presented by Dag

Svanæs (Norwegian University of Science and Technology).

A position paper, "Language Design and Informal Programmers" was accepted but not presented, as neither author Thomas Green nor alternate David Gilmore was able to attend.

Three invited discussants introduced themselves and their work, but did not present papers:

1. Bonnie A. Nardi, author of the end user programming classic *A Small Matter of Programming* [2]. (AT&T Labs West: Menlo Park CA.)
2. Jan Schultz, architect of PROMIS [3], a pioneering EUP system for medical records administration. (IDX Systems Corporation: Burlington VT.)
3. Greg Wolf, project leader of RICOH Silicon Valley's Open Source XML framework, the "Platform for Information Applications" [4]. (Ricoh Silicon Valley, Menlo Park, CA)

MORNING SESSION 1b: QUESTIONS ABOUT END USER PROGRAMMING AND HOW TO ANSWER THEM

The workshop began with a brainstorming exercise to identify topics of interest. All participants were asked to write either questions, or ways to answer previously asked questions, on 4x6" index cards, place them on posters, and then group them into categories. The remainder of this section gives some samples of each. We also chose 2 topic groups to discuss in the breakout sessions (SESSION 2.)

Theory

- How do we/should we distinguish programming from other computer work (find borderline cases; analyze essential features of programming; study cases.)
- What is Natural Programming; how can programming be made more natural?
- What kinds of EUP systems would be useful?
- How do we evaluate end user programming systems? According to what criteria?
- How do we coordinate our research efforts? (Example domains? Notations?)

Practice

- End user programmers vs. other informal programmers: how are the tools they need different?
- How can we create/support communities of informal programmers?
- What is the relationship and synergy between Participatory Design and end user programming?
- What Empirical Methods are relevant in the EUP design process? (Metaphor Analysis, PD)
- How can we build better debugging and testing environments for informal programmers?

- How are informal programming system "programs" kept up-to-date/in synch with the domain of their use?
- What is the role of training for EUP systems? What kinds of training would be useful?

Implementation

- How can AI techniques help EUP/PP?
- Problems with "advice" based on hidden information?
- How can we avoid 1000 domain-specific languages?
- Why are informal programmer systems hard to build?
- What are characteristics of a good underlying system to support EUP systems?
- The Web is the biggest new source of informal programmers today; how can we get the Web community interested in EUP?
- Can non-programmers describe behavior in XML?
- How can end users program tangible user interfaces (Furbie, Zowie, etc.)

MORNING SESSION 2: BREAKOUT SESSIONS (2)

Communities/Teaching&Learning/Domain Languages

The session began with a discussion on the different types of programming metaphors (visual vs. textual languages), in an effort to determine which is "best" for end user programmers. Or indeed, is one necessarily better than the other, and if not, how do we design a programming language that will be "good" for end users?

There is a common belief that visual languages (e.g. Lab-View) are easier to use and better suited to end users. However, there is no scientific evidence that visual is generally better or easier than text. Using a visual metaphor still entails understanding the characteristics of what you are generating. So, visual metaphors are not necessarily the answer, and can prove to be more difficult to use than text. Following patterns is not the optimal way either. An in-depth study of these different metaphors would help determine if one is really better than the other.

A community of users does not imply that all members have the same expectations of or level of expertise in a language. Indeed most communities are made up of a variety of different types of people who have some common goals, and use their particular expertise to help other members when necessary. Open source code may be the key to community computing. Open source code allows for different levels of use within the language. A community of users can thoroughly customize the environment to suit their needs. HTML is an example of a textual language that is being used successfully by various communities of end users. That HTML is an open-source-code language has greatly aided its success at being widely used by variety of non-professional programmers.

Not all the questions posed were answered.

- How do we represent the community knowledge of an office (e.g. who signs off on things?) computationally?

- How much do end users need to understand about the underlying architecture of a tool to be able to solve problems that arise?
- How do you create a common domain-specific language and avoid ending up with multiple domain-specific languages for the same domain?

End users and systems

Some of the participants described the end user systems and/or end users they are working with.

- A system written in XML that uses tags as behaviors. The goal is to allow people to specify domain specific systems in XML. Users must be able to change the system at a variety of levels, e.g., forms, company name, sequences etc. The code for this project is open source code as it allows people to really customize the system for themselves. There are two groups of users of the system. The first group is the domain people who create the behavior tags, and the second group is the people who use the tags to create domain-specific applications. There is a low bandwidth channel of communication between the two groups. An effort is being made to use these tags to create a wider channel of communication between the two groups. One aim is to make tag management a function within the community.
- A highly interactive system that allows users to access very complex medical information. Standard tags are used by end users.
- A system that deals with the finance, administration and clinical applications for a large medical institution.
- Biologists – A group of biologists not only use existing software but also need to write and adapt existing programs. The biologists do not want to spend time learning how to use tools. Giving them examples to work with is important. Participatory design methods are being used to find out how and why they would want to program.
- Teachers – The primary goal of this project is to determine what are the intrinsic and extrinsic motivations that make teachers willing to adopt a technology. A resource of network applications that support collaboration among students of science of different levels and who are at different locations is being developed. Teachers who enjoy programming are recruited as lead users. They create programs that can be used and modified by others. Currently four schools are participating in this project. There are many limits that need to be overcome in order to make it easy for the teachers to seize the new technology. Teachers need to be able to see potential rewards should they adopt the technology. Participatory design methods are also being used for this project.

Reflections on the discussion

The session ended with the following reflections on the discussion:

- Understand individuals' motivations for using a language, their social context and reward structure.
- Most organizations don't value time spent on learning.

- Communications within a community of users are more important than tools. Cooperation between different levels of users is necessary.
- If a software tool provides the user with positive feedback and rewards, i.e., they see that they can accomplish what they set out to do, end users are willing to put up with very unusable systems.
- End user programming encompasses a number of different levels (levels of expertise, levels of learning, levels of functionality and generalizability). All levels need to be examined. If the levels are well structured, it is easier to understand when problems have to go beyond one level to another.
- Open source is important to community efforts. It seems also to have an effect on programmer productivity, reuse, etc.

Natural Programming

"Natural Programming" is a project in Brad Myers' group at Carnegie Mellon University, and John Pane's thesis research. By studying users who have never programmed and applying HCI and Psychology of Programming principles, they are attempting to create general-purpose programming tools that are significantly easier to learn and use than traditional programming languages. Some findings of the project so far: untrained users

- Used a lot of text and few pictures -- used pictures early on for scenarios; then text to describe actions.
- Used an event-response model: "When Pac Man eats all the pellets, he goes to the next level."
- Used "all of" a set rather than iteration logic.
- Avoided complicated Boolean expressions. Instead they used mutually exclusive rules, or a general case with exceptions: "When the monsters touch Pac Man, he dies, unless he just ate a power pill."
- Used Boolean operators inappropriately: AND for OR (because in all languages, AND means 'add'); NOT was confusing.

Other points that came out in discussion:

- Users need metaphors drawn from prior experience. Current EUP environments use limited, domain-specific metaphors. Natural Programming seeks metaphors for general-purpose programming. Von Neumann computer concepts of time, states, iterations, and Boolean logic are *not* natural. Yet there *are* universal mechanisms in natural human languages. Similarly, programmers and non-programmers use *the same* expressions to during design. So it may be possible to find natural, general-purpose programming metaphors and mechanisms, as well.
- Objects with behavior are a natural way to program. However other object-oriented programming concepts like inheritance are not natural. End users don't like information hiding; they want to be able to see how everything works.
- AI tools such as agents, advice, and reversible debuggers may help users convert their intentions into precise pro-

grams. Examples include Chris Fry's ZetaLisp and Charles Rich's Programmer's Apprentice projects at MIT.

- It's hard to predict *a priori* what programming approaches will work with a particular group of users; participatory design is the best approach.

AFTERNOON SESSIONS

After lunch, the group chose to continue with whole-group discussions rather than small groups. The afternoon was divided into three sessions.

Group Discussion I

Starting with the topic of teaching programming, this became a discussion of problems end user programming encounters, and possible solutions.

Problems

Students nowadays have to be taught complicated Integrated Development Environments (IDEs) and Application Programming Interfaces (APIs.) Even HTML is becoming too complicated to share.

Many of the problems arise because so much software suffers from "featuritis". The cause is more fundamental than e.g. trade press articles with their checklists: it is the natural evolution of software products. The best ones start out simple, and lots of people can use them -- BASIC, HTML, etc. Yet as they become successful, more and more different communities of users adopt them. As features are added to meet all their diverse needs, the simple product becomes more and more complex, and requires a growing level of expertise to use. Eventually the learning curve becomes so steep that newcomers give up and desert it for a new tool that is simple again.

How can we get off this "wheel of reincarnation?" One solution may be software agents, just as in the real world we employ agents and experts to deal with complex domains on our behalf. (It was also pointed out that at some progress is being made already, because each "cycle" of languages includes higher-level primitives.)

A different problem is commercial software products that break their own applications with each new version. Successful end user programming systems are typically large programs in which both users and the developers who support them invest heavily. New versions of application tools force users intimately familiar with a product to relearn it. Developers find new language versions no longer compile *their* programs. Yet software vendors drive them to new versions by making file formats incompatible and cutting support for previous versions.

Solutions

Open source and open standards may be keys to achieving a reliable basis for end user programming. Both are driven by the user's interest, unlike commercial behavior. Closed-source programs require a leap of faith with each new ver-

sion; with open source, users and developers can see what they are getting.

Another approach to stability is to develop application frameworks independent of the commercial infrastructure. This is costly -- you have to build a lot of tools yourself in a lower-level language, and you can't benefit from improvements in new commercial software versions -- but it's often the right approach.

Finally, if we can't eliminate dependencies, we may invent software to manage them, i.e., the machine cross-compiler or converts programs and data.

A different problem introducing end user programming is social: the previous system's developers may not cooperate due to fear and prejudice. Education is needed.

Stagecast Creator demo

Allen Cypher demonstrated Stagecast Software's new end user programming product, *Creator*. This software, like Apple's *Cocoa* (*KidSim*), lets children build virtual worlds by creating characters and giving them characteristics and rules of behavior. Information and a trial version are at www.stagecast.com.

Group discussion II

The plan for the second afternoon discussion session was to focus on research methods and develop a research agenda. However the question, "how can we help end user programming get the use and recognition we think it deserves?"

It was suggested the CHI community views end user programming as a fringe area. One goal should be to persuade them EUP is possible, and CHI principles should be applied toward it.

There is also a view in the software industry that end users are incapable of significant programming. We need to demonstrate and/or make people more aware that PBE/PBD and other end user programming approaches make programming accessible to the general public. Spreadsheets, CAD systems, statistics packages, and industrial automation are areas where end user programming products have been successful.

Another suggestion to show the feasibility of EUP was to do more scientific studies to guide research in this area and establish principles. Examples that demonstrate the feasibility of programming by end users need to be collected and presented as case studies. Interesting researchers in performing empirical studies in this area is an important goal.

We need to raise the usability level of EUP tools to make programming accessible to the masses. Today's end users accept systems that are hard to use, and often accomplish great things in spite of them. Systems programmers tolerate difficult tools from "programmer macho". Neither get help from language designers obsessed with theoretical elegance.

Truly usable programming environments represent a huge market opportunity.

People at major companies already do EUP, even with very inadequate tools. It would be useful to find out what documentation exists to show who is doing EUP, and how much money could be saved with less effort and wider use from more usable tools.

Another economic argument: if a fraction of the many billion\$ spent on conventional programming every could be done more efficiently as EUP -- huge economic payoff.

To generate more interest in and awareness of EUP we need to show:

- What we mean by end user programming.
- Where it can be useful.
- Where it improves usability or productivity.
- Where it is more effective than hiring someone to write software for all these needs.

Research agenda

Methodology questions:

- To what extent should we build systems?
- Empirical studies (i.e., experiments, surveys, field studies) on EUP systems need to be conducted.
- What types of design methodologies (iterative, participatory design) employ for EUP systems?
- How do we learn from these systems?
- What would it take to get industry interested in EUP?

ABOUT THE AUTHORS

Howard Goodell (Workshop Organizer) is BS, MS Chemistry; a programmer for 20 years in controls and instrumentation software, and a doctoral candidate in Computer Science at U. Mass. Lowell in Lowell, MA.

Micrion Corporation
1 Corporation Way, Centennial Park
Peabody, Massachusetts 01960 USA
+1 978 538-6680
hgoodell@cs.uml.edu

Carol Traynor (Workshop Organizer) received her ScD in Computer Science from the University of Massachusetts Lowell, Lowell MA (1998). She is an Assistant Professor in the Computer Science Department at St. Anselm College, Manchester NH. Her research interests include end user programming. She has developed a PBD approach for Geographic Information Systems.

Computer Science Department
Saint Anselm College
100 St. Anselm Drive
Manchester NH 03102 USA
+1 603 656 6021
ctraynor@anselm.edu

Sarah Kuhn received her Ph.D. in Urban Studies and Planning from MIT, Cambridge, MA (1987). She is an Associate Professor in the Department of Regional, Economic and Social Development at U. Mass. Lowell in Lowell, MA. Research interests include the effects of computerization on work and software and engineering design. She is co-director of the graduate certificate program in Human-Computer Interaction <http://www.cs.uml.edu/~williams/hci/hcicer-tif.htm>.

Dept. Regional Economic&Social Development
University of Massachusetts Lowell
61 Wilder Street, O'Leary 500J
Lowell, MA 01854 USA
+1 978 934-2903
Sarah_Kuhn@uml.edu

Goodell, Kuhn and Traynor are members of the HCI Research Group in the Computer Science Department at the University of Massachusetts Lowell, Lowell, MA. <http://www.cs.uml.edu/~williams/hci/hcigroup.htm>.

David Maulsby received a Ph.D. in computer science from the University of Calgary (1995) for a thesis on interactive machine learning techniques for computer agents to learn repetitive tasks in office applications. The techniques enable end users to teach by examples mixed with verbal and gestural hints. Dr. Maulsby has also conducted research on agents and model-based user interface design at the MIT Media Laboratory and Stanford University. As an industrial consultant, he has created a commercial application of programming by demonstration, for RADSS Technologies, and has modeled workflow systems. Currently, he is a principal of 24C Group, an Internet software startup.

24C Group Inc.
300, 714 First Street South East
Calgary T2G 2G8 Canada
+1 (403) 264 7400
maulsby@24c.com

REFERENCES

1. Workshop Position Papers available at <http://www.cs.uml.edu/~hgoodell/EndUser/blend/papers/index.html>.
2. Nardi, B. (1993). A Small Matter of Programming: Perspectives on End User Computing. Cambridge <http://www.best.com/~nardi/ASmallMatter.html>, Cambridge: MIT Press.
3. Schultz, Jan, "A History of the PROMIS Technology: An Effective Human Interface," in *A History of Personal Workstations*, Adele Goldberg, ed. New York: ACM Press, 1988, pp. 439-488.
4. The Platform for Information Appliances <http://www.RiSource.org/>

