

End User Programming in an Industrial Research & Development Group

Howie Goodell

Doctoral Candidate, Univ. Massachusetts at Lowell

hgoodell@cs.uml.edu

Abstract

This paper presents partial results from a case study of a full application of end user programming in an industrial product Research & Development (R&D) organization. Scientists and engineers developing large industrial machines built all application software, with programmers (including the author) in a support role. In interviews, the project team and its managers described a sophisticated and complex integration of programming into their work activities, similar to those described in other end user programming environments.

1. Introduction

A product R&D center of a large technology company with which the author was affiliated developed a fully end-user programmed control system for complex production machines they designed. Its goal was to give scientists and engineers maximum flexibility for research, and full control over the product's behavior. This was considered important to overcome technical challenges and cope with rapid change in a new field.

Users ranged from technicians to post-doctorate scientists. Most had some exposure to a programming language -- one engineer had worked as a programmer -- but their primary work responsibilities lay elsewhere.

2. The end user programming environment

The machines performed processing steps in reaction chambers. Robot arms moved material between chambers in vacuum to avoid contamination. Operators controlled the machine in a GUI, by manual operations on screen icons and running automation programs.

Remarkably, the GUI included tools for end users to completely re-shape it for different machines:

- Create application-specific automation, interlock, and monitoring text programs – 29000 LOC on one machine alone – using a syntax-directed program editing and visualization environment analogous to the Cornell Program Synthesizer [1] (see Figure 1.)
- Construct GUI screens by graphically placing icons and connecting them to inputs and outputs (I/Os.)

- Graphically define hardware I/Os.

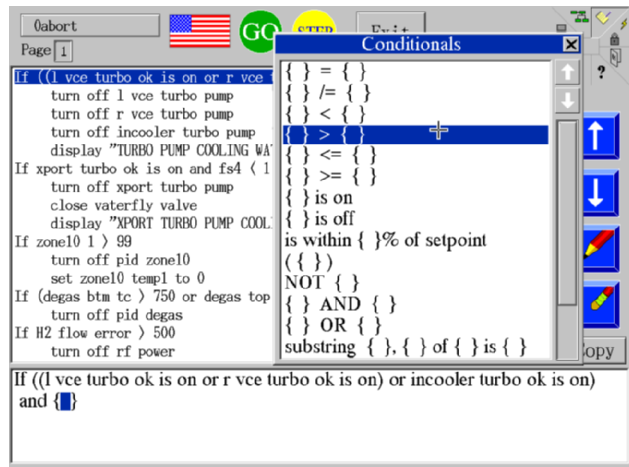


Figure 1: Syntax-directed editing

The tools could be invoked in seconds, whenever the operator wished. This encouraged an easy, interactive style of development. Discipline (for multi-million dollar equipment and potential safety hazards) was provided by user professionalism, etiquette, and rules.

3. Investigation

The author and his supervisor conducted 8 interviews with people who actually worked on the machines, and 3 with others related to the project. Most were conducted in person, in multiple hour-long sessions, which were tape recorded and transcribed, along with samples of their work.

Interviews were based on a questionnaire with a mix of brief and open-ended questions. Respondents were encouraged to use the questions as jumping-off points to explain details of their experiences, and a number did so at length. This paper presents a small sample of that outpouring, specifically related to the ways they incorporated programming into their R&D work process.

4. Results: Integrating programming into R&D work

Nardi and Miller [2] describe the complex patterns of interaction that arose among office workers applying the most successful end user programming tool, the spreadsheet. For example ([2] p. 530) spreadsheet users:

1. share programming expertise through exchanges of code;
2. transfer domain knowledge via spreadsheet...
3. debug spreadsheets cooperatively;
4. use spreadsheets for cooperative work ...and
5. train each other in new spreadsheet techniques.

In this environment, most of the collaboration seems to have been talking in advance about what the machine should do. After a discussion in a group meeting or between several researchers implementing a function, one of them would implement what they had decided. One young physicist with a knack for programming estimated he ended up writing over half the programs for his group. If the other users had been less sophisticated, he might have gained status as a "local developer" ([3] p. 104), but here he felt it neither raised nor lowered his standing; it was just a chore that had to be done.

It was common to ask for help with a new area or tricky programming technique. However two users actually writing a program together was unheard of; each program had one owner. This may have been influenced by the decentralized design of the machine: the person developing process chemistry in each reaction chamber naturally wrote its programs. Users never changed someone else's program unless it had a bug.

Copying one another's programs was a major mode of collaboration. After an author described a useful function in a group meeting, or an inquirer found an existing program that did what s/he wanted, they would copy it and change the I/O's and other details to fit their situation. This was usually accompanied by consultation with the author about the purpose of what was done. People programming a new machine copied code from older machines with similar functions. This was an opportunity to re-implement features more cleanly.

Like spreadsheet users for whom "Debugging is a task that is distributed across the group..." ([2] p. 540), this group decided on an "etiquette" that critical programs -- that could damage hardware or spoil a night's research -- be given to someone else to proofread. This practice sparked many discussions about how to do things better. These could be sophisticated: e.g. the author overheard two physicists debating deeply-nested conditional statements vs. multiple threads with semaphores.

Because the users became quite skilled, they only occasionally consulted system programmers for help on unfamiliar or tricky features. Their primary interaction was to report system bugs or ask for new features.

As with the spreadsheet users in [2], people developed very individual styles of relating to programming. Two of the managers were deeply involved in programming; one did none at all (but was enthusiastic about what it had done for his project); another averaged 15 minutes a day modifying text programs, but never built screens. Two senior physicists were notorious for having strongly contrasting programming styles, and frequently argued about the best way to implement a function (but sought one another's opinions frequently); others were pragmatists who wouldn't spend a minute more than necessary.

5. Future work:

Many of these users and managers stated strongly and in detail how being able to program the R&D machines had shortened their design cycle, made their research more effective, and lowered barriers to creative design. It may be hypothesized that R&D environments like this one, with frequent requirement changes and reliance on domain knowledge in multiple disciplines, are especially suitable for end user programming. If so, there is a potential market for control systems for machinery, data collection software, and other R&D support tools with end user programming capabilities. This intriguing possibility will require comparative studies -- perhaps occurring in the marketplace -- to demonstrate.

The author continues to analyze detailed interviews from this project in his doctoral research. He is also seeking other end user programming domains to test how hypotheses suggested by these results will generalize.

Acknowledgements:

The author thanks Nicolas Hofmeester for our many formative discussions and conducting 2 interviews, our coworkers who were generous with their time and experiences; and U. Mass. professors Robert Lechner, his advisor Marian Williams and co-advisor Sarah Kuhn, for their guidance and suggestions.

[1] T. Teitelbaum and T. Reps "The Cornell Program Synthesizer: A Syntax Directed Programming Environment", *Comm ACM* 24,9, Assoc. Computing Machinery, New York, New York, September 1981, pp. 563 -- 573.

[2] B. A. Nardi and J. R. Miller, "Twinkling lights and nested loops: distributed problem solving and spreadsheet development", *International Journal of Man-Machine Studies*, 34, 2, Academic Press, London, February 1991, pp. 528-551.

[3] B. A. Nardi, *A Small Matter of Programming*, MIT Press, Cambridge, Massachusetts, 1993.