GARDENERS AND GURUS: PATTERNS OF COOPERATION AMONG CAD USERS

Michelle Gantt
Bonnie A. Nardi

Hewlett-Packard Laboratories
Human-Computer Interaction Department
1501 Page Mill Road
Palo Alto, CA 94304
Internet: nardi@hplabs.hp.com
(415) 857-5121

ABSTRACT

We studied CAD system users to find out how they use the sophisticated customization and extension facilities offered by many CAD products. We found that users of varying levels of expertise collaborate to customize their CAD environments and to create programmatic extensions to their applications. Within a group of users, there is at least one local expert who provides support for other users. We call this person a *local developer*. The local developer is a fellow domain expert, not a professional programmer, outside technical consultant or MIS staff member. We found that in some CAD environments the support role has been formalized so that local developers are given official recognition, and time and resources to pursue local developer activities. In general, this formalization of the local developer role appears successful. We discuss the implications of our findings for work practices and for software design.

KEYWORDS: Cooperative work, CAD, end user programming.

INTRODUCTION

Recent empirical studies of end user computing have found a strong pattern of cooperative work among users of a variety of software systems, including spreadsheets, word processing programs, and Unix. Users with different levels of computer expertise and interest work together to customize their environments [5,15,16] and to program applications [23]. Nardi and Miller [23] identified a continuum of three kinds of users: end users, local developers, and professional programmers. End users have little or no programming education and tend to lack an intrinsic interest in computers; they are focused on their own domain interests. Local developers are domain experts who have acquired more advanced knowledge of computing, and in particular, knowledge of one or more specific software systems such as spreadsheets or CAD products. They serve as a resource for end users, training them and developing code for them. Programmers have a much broader, deeper knowledge of computing than local developers (e.g. knowledge of compilers, operating systems, languages, architectures, programming methodologies) acquired through professional training. Programmers contribute code to the programs of end users and local developers, and help them learn new things.[1] Mackay [15] and MacLean, Carter, Lovstrand and Moran [16] described a similar continuum of expertise among the users they studied.[2]

In this paper we describe patterns of cooperation among users of CAD systems. We chose an ethnographic approach so that we could discover what CAD users are actually doing, and what they think about what they are doing. Ethnography is well suited to studies of collaborative work as patterns of collaboration extend over many users, and are richly nuanced in a way that cannot be replicated in the artificial environment of the laboratory. In the everyday world, people work out ways to solve problems that accommodate and take advantage of the varying expertise and interests of group members. Such methods of work will not emerge under the short time frame and controlled conditions of laboratory experiments, but can be studied naturalistically.

The objectives of this paper are: (1) to document the nature of cooperative work among CAD users, as further empirical evidence of the pattern of collaborative end user computing practices reported in other studies; (2) to highlight the importance of the formalization of the local developer role that we discovered in some of the groups we studied; and (3) to discuss the implications of the cooperative nature of end user application development for work practices and software design.

Many recent studies have suggested that the introduction of computers into offices and factories does not correlate with increased productivity [3,10,12,30,35]. One key reason is that computers are not being used to their best advantage because we are still trying to discover just how to do that [3,10,12]. We believe that human-computer interaction researchers have a contribution to make to this discovery process as our con-

[1]There are end users who do not become local developers but who do become quite sophisticated computer users. They are in a minority. The local developer role would not have evolved if they were the common case.

[2]For *local developer* Mackay used the term *translator* and MacLean et al. the term *tinkerer*.

cerns encompass both work practices and technology, which must be considered together as we learn to utilize technology more effectively. Here we focus specifically on the cooperative nature of end user computing. We believe that a description of how users actually get applications written will inform our understanding of how to better organize and manage work practices, and how to better design computer products and research prototypes.

Our study of CAD use partly confirms what other studies have found, i.e. that end users collaborate with more experienced users to create applications. As described in detail in [5,15,16,23], the range of applications end users can create is much greater than it would be in the absence of collaboration for two reasons: end users are learning from more sophisticated users, and they are getting code from them. This sharing of knowledge and code among a community of cooperating users is a mainstay of end user computing.

However, our study found a significant difference in the pattern of collaboration among CAD users compared with users in the previously mentioned studies: the role of local developer has changed from an informal, ad hoc, unsupported position to a formal or semi-formal role in some organizations using CAD. Managers recognizing the benefits of having a local developer in place have begun to support the role. In the CAD world, local developers are taking on many of the functions usually assigned to a customer support organization or MIS department. Because of the tremendous backlog in such organizations, they may not deal with users' problems in a timely manner (or may not deal with them at all). In contrast, local developers serve smaller, more localized constituencies, and they already know the domain and associated problems. Formal support of local developers is exactly what Mackay [15], based on her study of Unix users, recommended: local developers should be given the time and resources to provide computer support to their fellow domain experts. In this paper we will describe the formalized and semi-formalized local developer roles in some of the organizations we studied, and evaluate the advantages and problems of acquiring local developers from the ranks of domain experts.

The "gardeners and gurus" of our paper are terms from the argot of one large corporation we studied where local developer roles have become formal or semi-formal (depending on the needs of the division or department within the corporation). Local developers supporting mechanical engineers in the corporation are called *gardeners*, and those in electrical engineering are *gurus*. The term gardener comes from the corporation's attempt to "grow productivity" by making a gardener responsible for nurturing fellow employees and providing support so that they can perform as effectively as possible. Gardeners and gurus are a special case of local developer. They are distinct from other local developers in that they are given recognition, time, and resources for pursuing local developer activities. In this paper, we will use "gardeners" as a convenient cover term for gardeners and gurus and whatever else they may be called in other organizations.[3] Where our comments apply to either local developers or gardeners, we use "local developer" as the more general term.

There are a number of studies of CAD use, including studies of CAD tasks from a cognitive perspective, the management of CAD users, and ways to use CAD systems more productively. Ullman et al. [33,34], Dillon and Sweeney [7], Cuomo and Sharit [6], and Pikaar [27] studied aspects of the cognitive processes of individual CAD users; for example, Dillon and Sweeney [7] compared the mechanical engineering design process with that of designers using traditional drawing board techniques. Petre and Green [26] compared the cognitive efficiency of graphical vs. textual notation in electrical engineering design. Sinclair, Siemieniuch and John [31] and Badham [1] described the work process, and design and drafting roles in CAD environments. Maver [20] and Krouse, Mills, Beckert and Dvorak [14] made recommendations for managing CAD environments to maximize user acceptance and productivity. Perzanowski [25] provided a primer of CAD specialization, describing how to write macros, and other aspects of CAD use. Sebborn described the use of customization for automating the work process [29]. Majchrzak, Chang, Barfield, Eberts and Salvendy [17] discussed the technical and user interface aspects of CAD, and the management of CAD users. Brooks and Wells [4] and Manske and Wolf [18] also studied management of CAD users, noting that formalizing the role of local developer can be a valuable support for end users, though they did not elaborate further. Graham [13] advocated developing a network of experts, each specializing in a different CAD tool, to increase productivity. We did not find any in-depth studies of interactions among different kinds of CAD users, or of how users cooperate in building applications.

CAD SYSTEMS

What exactly do we mean by CAD? Computer-aided design (CAD) began in the mid-1960s. Sutherland's SKETCHPAD [32], created at MIT in 1963, is credited with being the first (prototypical) CAD system, and with providing the impetus for future development [17]. Researchers at large automotive and aircraft companies such as General Motors, Lockheed, and Boeing began developing proprietary mainframe-based systems. Because these early systems were very expensive, their use was limited to those industries that could afford the necessary capital investment. The development of minicomputer technology made CAD commercially viable. In the late 1960s, electrical engineers began to use CAD as design and drafting aids for printed circuit board production. During the 1970s, CAD moved into architecture and mechanical and civil engineering. Today CAD has become a standard tool in all of these industries, and a wide variety of products are in use. (See [9,11,17] for the history of CAD development.) CAD systems

[3]A recent InfoWorld article entitled "Nurturing the Flock" describing various support arrangements for PC users featured a cartoon showing the PC Manager as a mother hen surrounded by chicks representing different kinds of users (novices hatching out of shells, demanding users fighting over worms, etc.) [28]. While we doubt that "Mother Hen" will emerge as the next term for local developer, the metaphor does highlight the problem of supporting users. Another use of the term gardening comes from the Soviet Union. In an article in the July 1, 1991 *San Jose Mercury News* it is reported that two Soviet management consultants advocate that managers must be persuaded to stop thinking like "mechanics" and start thinking like "gardeners" who want to cultivate new and improved organizations. It is interesting to see metaphors of the natural world (gardeners, chickens) pressed into service in technical environments.

run on workstations, PCs, and Macintoshes. CAD users have a relatively sophisticated computing environment (at least compared with many end users who are still in the world of standalone PCs and floppy disks). A typical CAD workstation includes a large color monitor, a drawing tablet with stylus and/or mouse, and often network access to other CAD users, printers and plotters, library file servers, and backup systems.

CAD packages are, at their most basic level, drawing and drafting aids. What makes them different from simple drawing packages is that behind each graphical representation is a dataset that enables the program to perform a variety of functions such as autodimensioning,[4] creating wireframes and solid models,[5] and generating parts lists.

Many CAD programs[6] are specialized by the manufacturer specifically for a particular industry such as architecture, electrical engineering, mechanical engineering, or site planning. Other programs, such as AutoCAD, have open environments that can be specialized by users to meet the needs of a wide variety of task domains.

Some CAD programs perform specific tasks within a particular domain. For example, within electrical engineering, there are several tasks: schematic capture, simulation, and physical layout/design. A few systems such as Chipbuster manage all of these processes, but usually each is handled by a separate software package.[7] There are also CAD systems that provide frameworks that integrate such task-specific applications; for example, ASG runs with AutoCAD to automate the entire architectural design process. Traditionally, many of the non-drawing processes (e.g. simulation) came under the rubric of computer-aided engineering, but the line between computer-aided design, engineering and manufacture (CAD, CAE and CAM) is becoming increasingly fuzzy as software companies integrate their tools to support the entire process, from design to manufacture.

METHODOLOGY

To study CAD use, we conducted in-depth interviews with 24 informants (21 users and 3 managers), collected and analyzed informants' CAD artifacts (printouts of macro programs, designs, etc.) and studied and used a CAD system (HP ME30). Informants were found through an informal process of referral. Interviews were tape-recorded in informants' offices or work settings.[8] A set of open-ended questions was asked of each informant in the course of the interview (see Appendix). We discussed the users' tasks and how they used CAD to accomplish them, and how CAD fit into the overall workflow. The order of the questions varied depending on the course of the conversation. Additional conversational leads were followed as they arose. Informants showed us examples of their work on-screen and in paper form. Approximately 325 pages of transcription were obtained from the interviews.

Our informants included architects, mechanical engineers, electrical engineers, and industrial designers. They came from seven companies ranging from a three-person architecture design office to Fortune 100 companies. Most informants had college degrees and all had at least two years of college. Most had been using CAD for at least five years. Informants' computer experience ranged from those who were completely self-taught to one engineer with a degree in computer science (and a degree in electrical engineering). Most informants had taken some formal programming or product classes. Many had also put in long hours studying on their own.

CAD software varies extensively by industry. In our study informants discussed many different CAD systems (all of which are named in footnote 6). The numerous software products discussed reflect the fact that often an individual uses several different products, and that users sometimes referred to products they had used on previous projects.

We introduce seven informants in some detail to give a more concrete flavor of CAD use and CAD users. Informant names used here are fictitious. Verbatim segments from the interviews of these informants will be given to illustrate aspects of CAD use.

- **Ben** is an electrical engineer/designer who works in the R & D laboratory of a large corporation. He has a BS in physics and an MS in electrical engineering. Ben is a self-taught programmer, with no formal programming education. He is a local developer and provides support to twelve people.
- **Steve** is an electrical engineer at a small computer company. He is working on the design of the company's product, which is not on the market yet. He has degrees in electrical engineering and computer science. Steve is a local developer.

[4]Autodimensioning is placing dimension lines and corresponding numerical measurements after two points are specified.

[5]A wireframe is a view of each surface of an object incorporated into one 3D view. A wireframe model contains no information on the content of an object that would be needed to compute, for example, volume, mass or stress. In solid modelling, such information is available because the inside or outside of an object can be determined [see 17].

[6]Product credit and trademark notification for the CAD products we refer to in this paper are given here: Abacus is a product of Hibbitt, Karlsson, Sorensen. Adobe Illustrator and PhotoShop are registered trademarks of Adobe Systems Inc. ADS, AutoCAD, AutoLISP and AutoShade are registered trademarks of Autodesk, Inc. ASG is a trademark of Archsoft Group. Cadence, SPICE and Verilog are trademarks of Cadence Design Systems. Calay is a product of Calay Systems Inc. Chipbuster is an internal Hewlett-Packard product. DCS and PCDS are trademarks of Hewlett-Packard. Design Architect is a trademark of Mentor Graphics Corporation. Dynaperspective is a trademark of Dynaware Corp. FlexiCAD is a trademark of Amiable Technologies, Inc. HILO is a trademark of Genrad. HP ME10 and HP ME30 are products of Hewlett-Packard. 20/20 is a registered trademark of Home Depot. KST, ArchT2 and ArchT2/3D are trademarks of Kativ Technologies Inc. Macintosh is a registered trademark of Apple Computer Inc. Mentor Graphics is a registered trademark of Mentor Graphics Corporation. ModelShop is a trademark of Paracomp. OrCAD is a registered trademark of OrCAD. Patran is a registered trademark of PDA Engineering. Pro/Engineer is a registered trademark of Parametric Technology Corp. Synopsys is a registered trademark of Synopsys Inc. Unigraphics is a trademark of McDonnell Douglas. Vellum is a trademark of Ashlar Inc. VersaCad is a registered trademark of VersaCad. Vivid is shareware. Unix is a trademark of AT&T.

[7]For example, Mentor's Design Architect is used for schematic capture, PCDS for printed circuit board layout, and Verilog for digital logic simulation of integrated circuits.

[8]The interviews were conducted by the first author.

- **Rick** is the CAD administrator (or "gardener") for a medium-sized architecture firm. His formal education includes BS and MS degrees in architecture. He has taken many programming classes. He and another system administrator support forty users, including ten CAD users.
- **James** and **Carol** are mechanical engineering production drafters for a large corporation. They support the production engineers in their division. Both have drafting degrees. James has taken many programming classes and is very interested in computers. He has taken on a semi-formal gardener position that includes overseeing the specializations done for a group of 100. In addition, James provides one-on-one support to five people in his group. Carol is an end user. She has no programming experience and does no specialization.
- **Mark** is the gardener for a medium-sized mechanical and industrial design firm. He began working as a mechanical engineering drafter fourteen years ago and was the first person to begin specializing the CAD system when his group began using computers. He has taken a few Unix classes but is primarily self-taught. He also maintains the office's other computers (used for databases and word processing) for forty-five engineers.
- **Warren** is an electrical engineer in a research and development group. He has taken many programming classes and studied extensively on his own. He is a full-time "guru" supporting seven engineers.

PATTERNS OF COOPERATION AMONG CAD USERS

In our study, we found that CAD system users follow the general pattern of collaborative customization and application development found in other studies [5,15,16,23]. CAD users cooperate both to customize their environments (as Mackay [15] found among Unix users) and to program applications (as Nardi and Miller [23] found among spreadsheet users). Two interesting differences from previous studies emerged: (1) the general level of computer sophistication is higher among CAD users such that professional programmers are rarely involved in the collaboration process; and (2) the role of the local developer has been formalized in some cases, with institutional recognition and support accorded the activities of local developers. We will describe general patterns of cooperation among CAD system users, and then discuss the implications of these patterns for supporting end user application development.

Customizing and Programming in CAD Systems

To better explain how CAD users cooperate, we briefly describe the kinds of specializations CAD users make to customize and extend their systems.

Simple customizations include changing the color of a menu, the location of a menu item, line widths and colors, drawing size, and so on. Such customizations are made by editing parameters in macros.[9] A more elaborate customization might involve writing a new macro. For example, the user might want to create a circle with the center lines automatically drawn in, and add the new circle to a menu. The user would write a macro, assembling the appropriate circle-creating commands and line-creating commands, specifying certain parameters, and then linking them to a menu slot.

The most complex form of specialization requires writing programs in languages such as C or AutoLISP,[10] writing Unix shell scripts, or writing programs in the complex macro languages of some products. Such programming is required, for example, to perform calculations needed for CAD applications (e.g. calculating part placement) or to link a CAD application to other programs (for example, users might need enhanced plotting utilities or links to simulation or database programs).

CAD macros may be called from within a programming language such as AutoLISP. For example, in an architectural application, when a symbol (such as a door or window) is selected from a menu, an AutoLISP function may call a macro that queries the user for values for height and width, which are then used to size the symbol before the user places it in the drawing.

End Users

Of the seven end users in our study, four had done no customization or programming. One was not allowed to do any, as his manager insisted on maintaining established conventions. The other three preferred not to specialize, and because they had local developers or gardeners supporting them, they did not have to. Three end users made simple specializations, as described below.

An end user's first efforts at specialization usually involve customizing the CAD environment by creating keyboard macros and/or changing parameter values within existing macros. These macros may come from many sources: from the software package itself, from macro sets already created for a particular division, department, or site within a corporation, from macros created by fellow users (other end users as well as local developers), and from macros published in magazines and electronic bulletin boards.

Like spreadsheet end users [23], CAD end users generally avoid manuals. They work by: (1) editing existing macros; (2) using existing macros as templates; (3) learning new program features by asking other end users and local developers how to do new things; and (4) asking others, especially local developers, for help in debugging macros.

As was found with spreadsheet users [23], end users of CAD systems tend to be focused on the domain and the task at hand, rather than showing an intrinsic interest in computers. Carol, one of the users who did not do any specializing, explains her focus on drafting skills:

[9]Many CAD "macro" languages are programming languages with features such as loops and conditionals. The complexity of the macro languages varies by product; some are quite simple, and others approach the functionality of conventional programming languages. Macros are organized into text files as with conventional programming languages.

[10]AutoLISP is a subset of XLISP and Common LISP, with some additional functions to support design tasks.

> **Carol**: And it does seem like probably just because of different personalities, we all sort of have our area of expertise that when a certain job comes in, [we can say], "Oh, Harry would be good for this" because technical illustration is really his high point...
>
> **Interviewer**: What's yours?
>
> **Carol**: Ummm, I think I'm very good at drafting skills; I know what those views should be, where they should be placed, and I think I'm very good at being a checker on all the specs on the drawing – that when this goes into production, we won't have to scrap parts, we won't have to bring it back for revision after we've gone through with the final tooth and comb. And that's what I enjoy about it... I tackle it like a puzzle, I want to comb everything out and cover every aspect of it and then know that... everything's perfect. I like that part. And like I said, James really likes the computer side of it and, "What can I get this thing to do?"
>
> **Interviewer**: Yeah, "What can I make it do next!" Well, and by not having to worry about the integrity of your lines, you can spend more time checking the integrity of the specs. ... So...
>
> **Carol**: So the content... is the main focus.

Rick, a gardener, summarized his role in terms of end users' focus on "content": "That's why I'm here, because I know that when these guys are designing, they just want to design. They don't want to have to look at a manual, they don't want to have to... get into any of that."

Local Developers

In the CAD world, local developers write the macros, programs, and shell scripts that are needed for many applications, but that are beyond the scope of end users' interests and abilities. End users rely on the output of local developers and incorporate their macro, program, and shell script files into their own environments.

Local developers also help end users write, complete, and debug macros. For example, two engineers at a mechanical engineering consulting firm wanted a macro that could create a parabola, a function that the firm's CAD system did not include. They did not know how to write macros, and the local developer did not know the math behind parabola creation. They worked together, each adding their particular expertise, to create the macro for drawing parabolas:

> **Mark**: Two women were dealing with a lens for a lamp and they wanted to... be able to define a parabola easily, and they were doing it somewhat laboriously. As it turned out, the ellipse command that is the standard ME10 command is a very complex macro that has you defining the major axis, the minor axis, and then it uses the spline command and repeats with all of the points. It's sort of a left-brain, right-brain thing. I can't explain it. I understand [it] when I'm writing it. So I was able to take that and simplify it greatly and make a parabola macro out of it... So, by just looking in a standard engineering book, we were able to take the math and I knew the [macro] language, they knew the math, and they just told me how we were supposed to manipulate the numbers that we got out of it.

In our study, the eight local developers (informal local developers, not gardeners) among our informants evolved from end users; they were not hired from the outside. They grew into the position because they started specializing on their own initiative, usually out of frustration with the existing software, and because they got interested in seeing how far they could push the software.

Ben explains how he got to be a local developer:

> **Interviewer**: You mentioned there were two reasons why you customize so much: one is because it's so easy [using Chipbuster], and I don't think we actually got to the other one...
>
> **Ben**: The other was aptitude, interest, frustration. I know it can be done, therefore I must.
>
> **Interviewer**: So how did you learn to write [macros] – the manuals?
>
> **Ben**: There's a tutorial manual, but mostly it's by learning from examples of how do people do similar things.... Very often what I want to do is tweak one of these existing commands so it does something a little differently...

Local developers evince a higher level of interest than end users in acquiring computer expertise. They are willing to wrestle with software and with manuals to achieve their aim:

> **Rick**: ... [Local developers] tend to be hackers: "I don't read manuals, I just start a program and say, 'Ah, let's see what it does! Oh, I have this other package that does the same thing.'" Then you look and see how it does it, and then you get stumped, and then you go look in the manual for reference. And you know in an afternoon you can figure out everything about a package. That's usually what I do.

For macro writing, local developers generally use manuals as a reference when they get stuck, or if they are the first one at a site creating specializations. Many informants spoke ill of manuals, as though they were necessary evils (we return to manuals in the Discussion section). Local developers doing complex programming in C, AutoLISP, or other languages use manuals for learning those languages – indeed there seems little alternative, as it is not possible in conventional programming to rely on editing existing code and/or pure experimentation, as one can for simpler specializations.

These findings about the activities and interests of local developers in CAD are consistent with reports of local developers in other domains [15,16,23]. The main difference we found in CAD is that local developers show considerably more computer sophistication than their spreadsheet counterparts (and, we believe, the users in [15,16]). While local developers of spreadsheets generally work within the macro language (and other specialized aspects of spreadsheets such as facilities to create fancy charts and graphs, or new formats for presenting cell values), local developers in CAD go beyond that level by writing in general programming languages or the complex macro languages of some CAD products, creating shell scripts, and becoming knowledgeable about operating systems. When spreadsheet users need to link to other programs, write complex macros, or do similarly advanced things, they typically call in professional programmers (e.g. from a customer support organization). In the CAD world, a class of users who are not professional programmers, but who come from the ranks of domain experts, have taken on such tasks themselves. It is as though the level of computer sophistication distributed across the different kinds of users has shifted up a level, popping programmers off the end.[11] In the Discussion section we explore the implications of this situation.

Gardeners and Gurus: A Special Case of Local Developer

One of the main findings of our study is that the informal position of local developer has evolved into a formal or semi-formal position in some organizations. In three of the seven companies we studied, gardeners were present. We interviewed six gardeners. As managers begin to notice the time and effort being expended by local developers – and to notice the benefits of local developer activities – they realize that formalizing the position can increase user productivity. In the semi-formal situation, local developers continue with their local developer tasks (as well as their regular duties), but they are at least now given recognition, appreciation, and possibly resources for the functions being performed. When the local developer position becomes fully formalized, the local developer is given a new job title, and time and resources to pursue local developer activities, usually full-time. Managers benefit from recognizing the local developer role in that there is now someone who can officially be relied upon to help end users, and to maintain standardization of the macros and programs they use.

What exactly do gardeners do that is different from informal local developers? In addition to performing traditional local developer duties, gardeners are responsible for writing and disseminating standard macros and programs at the corporate, division, or department level, and for researching and providing new tools to end users. The macros and programs they write may originate from their own observations of what is needed, or they may be created in response to user requests for certain capabilities, or requests from management. Sometimes a gardener sees a macro or program a user has written that looks useful for the whole group. The gardener takes the user's file, tests the code, modifies it if necessary, and then disseminates it to the rest of the group. Gardeners are always on the lookout for tools to enhance the group's productivity.

[11]Professional programmers are used to write proprietary CAD systems. We did not study any such systems.

James, a drafter who has become a gardener, describes a gardener's activities:

> **James:** The gardener has to have a good working knowledge of Unix... Because your IT [Information Technology] group is only going to know how to do the administration-type part. They're not going to know the ME30 part. Your end users are only going to know the ME30 part and they're not going to know the systems part. So a gardener is like a cross between both worlds. And he's got to be able to communicate what IT is trying to do with the standardized configurations and hardware ordering, and he's also got to be able to speak the language of the end user who's sitting there saying, "I'm unproductive and I need to be productive real fast." ... So, it's a juggling act...

As James's comments show, a gardener has both domain and computer knowledge. James has educated himself through programming classes and self-study to the point where he can handle the Unix and HP ME30 problems effectively. He has the drafting background to understand domain experts' problems and frustrations. James is effective with two constituencies – the systems administrators and the domain experts. Given appropriate tools such as CAD products, it seems that being a good gardener is generally easier if one starts on the domain side and acquires the necessary computer expertise, rather than starting on the computer side and trying to acquire a working knowledge of a domain. A gardener who is a domain expert need learn only a subset of computer science – that which applies to the specific jobs to be done within a domain. It would be more difficult for a computer specialist to acquire the understandings that accrue over time to a domain expert. Such understandings involving workflow, group work practices, and the problem areas and frustrations of the job are important to the gardener role, and are acquired over time, through the experience of actually doing the job.

In our study we found that informal local developers may coexist with gardeners in large organizations. In this situation gardeners take on the standardization and system maintenance tasks, and local developers tend to provide the one-on-one support to end users. There is potential here for a natural progression for some local developers to eventually become gardeners when gardeners leave or move within the organization.

DISCUSSION

Forester [10] notes that there is a "new wave of skepticism" regarding the productivity benefits of automation. Many studies suggest that the introduction of computers into offices and factories has not generally correlated with increased productivity ([3,10,12,30,35]). Forester calls this the "productivity puzzle." There are many complex reasons why automation has not produced the expected productivity gains,[12] but a key piece of the puzzle is that our work practices and technology have not yet

[12]The puzzle has many pieces, including some that have nothing whatsoever to do with technology. Productivity is difficult to measure. A net decline in productivity can occur when countervailing forces (e.g. the need for increased legal and personnel staff to monitor government regulations and employee entitlements) depress productivity more than automation, which is contributing

evolved to the point where we can take full advantage of the potential benefits of automation [3,12]. Given the short history of computing this is not surprising, but it is time for us to take a careful look at the insufficiently evolved social and technical bases of our present computing practices, and try to see where we can do better. Our findings on the cooperative nature of CAD use have practical implications in terms of both work practices and software design. We will argue that: (1) despite a few problems, managers will be well served by growing local developers into gardeners; and (2) software companies designing new products or enhancing existing products should consider how software is actually used – by groups of cooperating users with different levels of computer expertise and interest.

Cultivating Gardeners

Our study supports Mackay's contention [15] that the activities of local developers should be recognized and promoted. In the organizations that we studied in which gardeners have become formal or semi-formal positions, our assessment is that the benefits far outweigh the costs. End users are comfortable with CAD software because they are assured assistance in all aspects of CAD use – whether they are writing macros, learning new tools, or keeping abreast of the latest developments in CAD. Gardeners – who enjoy tinkering with computers – are given official leave to do so. Their communication talents are engaged as they provide an important bridge between system administrators on the one side and domain experts on the other, as well as communicating with users as they help them in debugging, learning new capabilities, and so forth. A gardener can save time and money by making it unnecessary for users to spend their time re-inventing the wheel (creating redundant macros and programs); instead, the gardener offers standard versions of these resources to the entire group. Employees can be more productive because they are concentrating on their domain-related tasks. Establishing a gardener creates a finer-grained division of labor among software users that increases efficiency and motivation because it is firmly grounded in users' interests and abilities.

An important benefit of gardening is that managers can feel confident that the standardization and integrity of the macros, programs, and data used by their staffs will be maintained. Many managers and MIS personnel are made somewhat nervous by the whole notion of end user computing because the potential for chaos is real: data can disappear, users may waste time managing their systems instead of doing their work, and esoteric, personalistic specialization can reduce the utility of the programs developed by end users (see [24]). With a gardener in place these concerns can be dealt with. It is the gardener's job to maintain standards, and to offer users standard programs that they can use to get their work done. A gardener is someone between management and users, and is trusted by both. Because gardeners are fellow domain experts and not outsiders who lack understanding of the everyday patterns of work and group interaction, gardeners are more likely to be effective in dealing with the group's concerns. They can in fact anticipate these concerns and handle them proactively in a way that is impossible for outsiders such as systems analysts or MIS personnel.

A problem with formalizing the local developer role that we found is that not all managers recognize the benefits of having a "productivity" person on board. They may feel that this is a waste of a person who could be working to "contribute to the product." Where a higher level of management mandates a gardener, the lower-level manager may feel that he or she is less effective and cannot utilize available resources in the best possible way (see [4]). When this happens, at a minimum it would seem useful for managers to evaluate whether a project really does have enough resources. It may be that a manager has not thought through the benefits of including a productivity person in the group, or that the manager is truly in a situation of insufficient resources. Adding a gardener to a group will involve some kind of economic analysis to determine just when the shifting of resources really makes sense. An indication of the need for a gardener is the situation in which a local developer is devoting considerable time to group support activities, without official recognition and support. Experimentation with the gardener role will undoubtedly be necessary to make it fit the needs of the individual group. Some groups may feel the need for a full-time, fully formal gardener, while others can get along with a semi-formal, part-time person.

Another problem with gardening is that there is not always a person with the right combination of technical and social skills available to assume the job. A person who is technically skilled but uninterested in intensive interpersonal interaction may not have much of a green thumb when it comes to helping other users. By the same token, a person without sufficient technical skill would not be an effective gardener. In our study we found a somewhat reluctant local developer, Steve, who typifies at least part of the problem of finding the right mix of skills. Steve is a highly technically skilled electrical engineer. He and another engineer work on a project where they are creating complex simulations. Some of the other teams within the company are doing similar simulations, and, instead of learning how to program the simulation themselves, they ask Steve and his partner to help them. Steve's ambivalence about acting in a support role comes through in the following only half-humorous exchange:

> **Interviewer:** So you guys are sort of the experts?
>
> **Steve:** We're supposed to be, but...
>
> **Interviewer:** (laughing) But you're not really? Well, do people come and ask you for help?
>
> **Steve:** (laughing) Well, the thing is, I'm no smarter than any of those guys and... we try not to answer their questions.
>
> **Interviewer:** So you don't like to share what you've done?
>
> **Steve:** No, no, we will help. It depends on the time...

How do organizations cultivate gardeners? The right mix of skills is important, as the above example illustrates. A gardener cannot be randomly chosen out of a group of users; there must be genuine interest and enjoyment in assisting less

to productivity, can increase it. But even taking these factors in account, we do seem to be underutilizing our technology [3].

knowledgeable users on the one hand, and the desire to learn one's way around the operating system and a programming language, on the other hand. Brooks and Wells [4] noted that when users are learning a new system in a training class, often one person stands out in interest and ability, and can be identified fairly early as a potential local expert. More generally speaking, their point is well taken: rather than selecting someone to be a gardener at the introduction of a new system, it is better to wait until users have had some exposure to the system, looking for those who gravitate naturally toward its use.

The results of our study indicate that another factor that contributes to gardeners' effectiveness is that they come from the rank and file: they know the domain, the users, the frustrations and problems. The need to expend great effort translating domain knowledge to computer experts is avoided. Gardeners who are domain experts have "been there before" - and, as we saw with James, they show concern and empathy for their users. They know how it feels to be frustrated and unproductive because of software or hardware limitations. This is a considerable advantage: CAD users work within a very challenging environment - design and manufacturing processes are changing, deadlines are getting shorter as managers scramble to reduce time to market [2], and the software itself is quite sophisticated, and constantly changing.

All the gardeners in our study had started out as domain expert/end users. Manske and Wolf [18] reported that in the organizations in Germany that they studied, the CAD administrator was a mechanical drafter with computer expertise, but not design expertise, hired from the outside. They did not provide an evaluation of how well this arrangement works, but it would be interesting to know more about the exact functions of such CAD administrators, and to compare them with their gardener counterparts.

A problem that Mackay reported in her study of Unix customization [15] was that the local developers (or translators as she called them) often were not as knowledgeable about computers as would have been desirable, and as a result, they sometimes distributed buggy code throughout the organization. We did not find this to be a problem among the CAD local developers/gardeners we studied because their level of computer expertise was high, and they were able to meet the technical challenges. While CAD users may typically start out with an advantage in having a good "technical" background, users in other fields can strive to attain a high level of computer competence through training and studying on their own. (Indeed, many of our CAD users had taken programming and product classes, and had devoted a great deal of time to self-study.) Cultivating gardeners should involve making the time and money available for such training and study. More generally, solving the problem of distributing quality code means supporting the local developer role in a formal way, to ensure that adequate time is devoted to testing and debugging code to be given out to the group.

If an organization can manage to train gardeners to a high enough standard, the need for professional programmers largely disappears, at least with respect to the well-designed commercially available products that support end user computing, such as CAD systems. In our study we found no instance of users resorting to programmers for assistance in specializing their CAD software. When macros and programs are written by gardeners, it is with group use in mind (so idiosyncratic code is not produced), and with a good command of the domain and the end users' concerns. In Mackay's study, the Unix customizations were written by professional programmers for their own use, and the "translating" needed before end users could make use of the customizations was considerable. Of course that was a different setting, and the comparison cannot be stretched too far,[13] but the point is that gardeners can produce code from the outset that is intended for group use, and, most importantly, with a very clear picture in mind of the group's needs and preferences.

While we believe that a major strength of gardeners is their origin as domain experts, we can also see a potential problem in this arrangement. Over time, full-time gardeners may lose touch with the domain side of gardening as their activities come to be defined purely in terms of support. The most effective gardeners will make a special effort to keep up with advances in their field, and to be cognizant of the changing work practices in which they themselves are not directly participating. Some gardeners may want to rotate out of gardening periodically to renew and update their domain knowledge.

Another of Mackay's recommendations was that local developers and programmers should supply end users with extensive examples so that they do not have to start from scratch [15]. As we have described, the users in our study typically had a rich body of macros from which to choose to begin their customizations and programs. This worked well, and we think that an important aspect of gardening is providing such examples. However, two qualifications to an enthusiasm for examples are in order. First, examples seem to work best when they are in a language that end users can understand well, e.g. for CAD end users, the macro language. It will not do to overload people with elaborate examples in complex languages - the presence of examples alone is not enough (see [8]). Second, we noticed that what the less experienced CAD users were doing with examples was simply changing parameter values. This suggests that examples should be set up so that much can be accomplished by making such changes. The use of examples - at least for beginning users - becomes a simple form-filling exercise, not an attempt to replicate functionality involving complicated datatypes, control structures, and so on. MacLean et al. [16] followed this rule to good effect in their "Buttons" prototype.

Cooperative Work and Software Design

There are lessons to be learned from the study of collaboration among CAD users that can be applied to software design as well as to work practices. In doing this study, we were surprised at the level of computer sophistication attained by the local developers and gardeners we talked to - which was generally higher than that of the spreadsheet local developers we studied [23], as we have described. It is important for software designers to think about two things: (1) the continuum of

[13]No criticism of the programmers in Mackay's study is intended in any way; they had no charter to create end user customizations.

expertise of a group of users who will cooperate in creating applications; and (2) the endpoints of the continuum.[14] We hear many exhortations to "Know the user!" but there is not a *single* user to know – there is a *community of cooperating users lying within a particular range of computer expertise.* Of all the things software designers might want to know about users, understanding their tasks, and their levels of interest and expertise with computers should be among the highest priorities.

Software designers need to find out, for a given product, where the continuum of computer expertise starts and ends, and then incorporate into the product a range of capabilities that takes advantage of the range of users arrayed along the continuum – just as today's most advanced CAD systems provide the basic drawing and drafting capabilities, a macro language, and a programming language such as AutoLISP.[15] Software products should have a carefully designed set of capabilities targeted specifically for end users, such as the drawing and simple macro editing/writing capabilities in CAD systems, or formula writing capability in spreadsheets. Then, moving along the continuum, more advanced users – local developers, gardeners and sometimes programmers – need to be supported with more sophisticated functionality for their own use, and because they will be the conduit to end users for the product's advanced capabilities.

Organizing functionality around different kinds of cooperating users makes the range of things end users can do with software products much greater. It also acknowledges that some end users will eventually attain more expertise with a program as they become more familiar with it (whether they come to act in the supporting role of local developer or not) and will want to be able to learn more sophisticated capabilities. Some approaches to end user computing, such as programming-by-example [19,21] and user modeling [36], assume that not only do end users remain rather deficient, but that they work in isolation, without a community of cooperating users. This is a limited view of end users, and will ultimately mean inhibiting their growth and the kinds of applications they can create. Three of our users brought this point up in their interviews spontaneously. (We did not ask about it; it was a topic they introduced because they thought we would be interested.) Rick described three levels of users who should be supported – in his terms: "the idiot level, the full-fledged program level, and the power user level." Warren, whose CAD system (for electrical engineering) offers few facilities for specialization, expressed dismay at the limitations of his system compared with another more sophisticated system:

> **Warren**: I'm very jealous because they have this incredible flexible ability to change and do anything. One of the problems with people who write tools is that they assume ... they try to protect the users too much. And they don't recognize that there's different levels of users. You can have beginners, general users and experts, and for the expert user, a lot of places will hire somebody to be an expert user just to [do] the customization, to do all the neat productivity things that make such a difference. But you have to have that flexibility to start with. ... I find myself with most of the PC [printed circuit] tools, there's not much I can do with them.

The upper end of the continuum of expertise is perhaps trickiest to assess – just how far can users who have a strong interest in computers but who are are not professional programmers go? It seems reasonable for at least some CAD programs to support a programming language such as AutoLISP, while such a language might not be suitable for other domains where users are less familiar with computer technology. Of course to some extent what we have here is a moving target, but we believe it is worth the effort to do a detailed assessment of potential users' computer abilities and interests before launching a product.

Another way that software companies can provide better products is to take advantage of local developers in planning manuals. Everyone complains about manuals; end users avoid them, and local developers use them when they "get stuck." We believe that manuals need to be radically re-thought, and that local developers ought to be part of that process – they should be hired by software companies as consultants in the planning and reviewing of manuals. A set of manuals geared specifically toward local developers, who are the ones apparently making the most use of them, would be useful. The standard reference manual format does not fit the bill – at least the complaints suggest that it does not. Our users mentioned the importance of well worked examples as being critical; such examples are often in short supply in reference manuals. Because local developers report that they use manuals when they "get stuck," much more extensive indexing of manuals would seem to be a promising solution. Given the complexity of today's software products, there is a huge number of highly specific places one can get stuck, and they often fall between the cracks of the relatively gross categories that comprise the average index.

SUMMARY

We described cooperative work among CAD system users, focusing on the different activities of end users, local developers and gardeners. We found that in some organizations the local developer role has evolved into the formal or semi-formal role of "gardener," obviating the need for professional programmers. The advantages and problems of the gardener role were discussed. Suggestions for improving cooperative work with software systems were given with respect to managing and supporting users, and for product design. We conclude that more effective use of software systems will be made when managers cultivate gardeners, and when software design efforts take into account the patterns of cooperation among different kinds of users working together to create applications.

[14]The notion of a continuum of cooperating users may not apply to every kind of software product, but it does apply to a large class of products. Cooperation among users is increasing, not decreasing. Even home users join user groups, make use of telephone support, read bulletin boards, ask friends for help, etc.

[15]Not every CAD system need provide the more advanced capabilities of course, but there clearly is a demand for them as people attempt applications of ever-increasing complexity.

APPENDIX: LIST OF INTERVIEW QUESTIONS

1. What is your job/position?
2. What is your educational and job-related background?
3. What is the work process/flow and where does CAD fit in?
4. Do you share your work with others in your group? If so, how ? (Do you share CAD files, paper drawings, etc.?)
5. What CAD system is used? For what function?
6. Has your system been customized? If so, who does it and what specific customizations have been done?
7. How did you learn to customize (if applicable)?
8. Is there a CAD expert in your group? If so, how did that person get to be the expert?
9. What do you like/dislike about the system you use?
10. What would make it easier to use (manuals, better organized program, etc.)? What other features would you like to see?

ACKNOWLEDGEMENTS

Many thanks to Janice Bradford, John Drabik, Chuck Habib, Dan Hirano, Russell Sanchez, Debbie Schultz, Celeste Welch, Julie Wilker, Jonathan Yen, and Pete Zivkov for their help in providing valuable information and names of possible informants. Jeff Johnson, Nancy Kendzierski, Jim Miller, Andreas Paepcke, and Craig Zarmer gave helpful comments on earlier drafts of this paper. We are very grateful to our informants who willingly donated their time to this study and shared their work and thoughts with us.

REFERENCES

1. Badham, R. Computer-aided design, work organization and the integrated factory. *IEEE Transactions on Engineering Management* 36, 3 (1989), 216-226.
2. Berardinis, L., Dibble, M., Dvorak, P., and Rouse, N. CAD/CAM industry report. *Machine Design* (May 23, 1991), 47-58.
3. Bowen, W. The puny payoff from office computers. In *Computers in the Human Context.* T. Forester, Ed. Basil Blackwell, New York, 1989, 267-271.
4. Brooks, L., and Wells, C. Role conflict in design supervision. *IEEE Transactions on Engineering Management* 36, 4 (1989), 271-281.
5. Clement, A. Cooperative support for computer work: A social perspective on the empowering of end users. *Proceedings CSCW'90.* (Los Angeles, 5-7 October, 1990), 223-236.
6. Cuomo, D., and Sharit, J. A study of human performance of computer-aided architectural design. *International Journal of Human-Computer Interaction* 1, 1 (1989), 69-107.
7. Dillon, A., and Sweeney, M. The application of cognitive psychology to CAD. *People and Computers IV: Proceedings of the Fourth Conference of the British Computer Society Human-Computer Interaction Specialist Group.* (University of Manchester, 5-9 September, 1988), 477-488.
8. DiSessa, A. A principled design for an integrated computational environment. *Human-Computer Interaction* 1, (1985), 1-47.
9. Encarnacao, J., and Schlechtendahl, E. *Computer Aided Design: Fundamentals and System Architectures.* Springer-Verlag, Berlin, 1983.
10. Forester, T. *Computers in the Human Context.* Basil Blackwell, New York, 1989.
11. Foundyller, C. *CAD/CAM, CAE: The Contemporary Technology.* Daratech Associates, Cambridge, Mass., 1984.
12. Franke, R. Technological revolution and productivity decline: The case of US banks. In *Computers in the Human Context.* T. Forester, Ed. Basil Blackwell, New York, 1989, 281-290.
13. Graham, B. Applying software tools to enhance engineering group productivity. *Proceedings of the Fifth Annual Applied Power Electronics Conference and Exposition.* (Los Angeles, 11-16 March, 1990), 612-618.
14. Krouse, J., Mills, R., Beckert, B., and Dvorak, P. CAD/CAM planning: 1990 - Managing people and the technology. *Industry Week* 239, 13 (1990), CC4-CC10.
15. Mackay, W. (1990). Patterns of sharing customizable software. *Proceedings CSCW'90.* (Los Angeles, 7-10 October, 1990), 209-221.
16. MacLean, A., Carter, K., Lovstrand, L., and Moran, T. User-tailorable systems: Pressing the issues with buttons. *Proceedings, CHI'90.* (Seattle, 1-5 April, 1990), 175-182.
17. Majchrzak, A., Chang, T., Barfield, W., Eberts, R., and Salvendy, G. *Human Aspects of Computer-Aided Design.* Taylor and Francis, Philadelphia, 1987.
18. Manske, F., and Wolf, H. Design work in change: Social conditions and results of CAD use in mechanical engineering. *IEEE Transactions on Engineering Management* 36, 4 (1989), 282-292.
19. Maulsby, D., Witten, I., and Kittlitz, K. Metamouse: Specifying graphical procedures by example. *Computer Graphics* 23 (1989), 127-136.
20. Maver, T. Social impacts of computer-aided architectural design. *Design Studies* 7, 4 (1986), 178-184.
21. Myers, B. Text formatting by demonstration. *Proceedings CHI'91.* (New Orleans, 27 April - 2 May, 1991), 251-256.
22. Nardi, B., and Miller, J. The spreadsheet interface: A basis for end user programming. *Proceedings Interact'90.* (Cambridge, England, 27-31 August, 1990), 977-983.

23. Nardi, B., and Miller, J. Twinkling lights and nested loops: Distributed problem solving and spreadsheet development. *International Journal of Man-Machine Studies* 34, (1991), 161-184. (Reprinted in *Computer Supported Cooperative Work and Groupware*, S. Greenberg, ed. Academic Press, London, 1991.)

24. Panko, R. *End User Computing: Management, Applications, and Technology*. John Wiley and Sons, New York, 1988.

25. Perzanowski, P. Scheduling CAD productivity. *AACE Transactions* (1991), I.1.1-I.1.5.

26. Petre, M., and Green, T.R.G. Requirements of graphical notations for professional users: Electronics CAD systems as a case study. In press. *Le Travail Humain* (1991).

27. Pikaar, R. Situation analysis of design tasks for CAD systems. *Behaviour and Information Technology* 8, 3 (1989), 191-206.

28. Raths, D. Nurturing the flock: As the PC population grows, so does the burden on support staff. *InfoWorld*, 19 August, 1991, 38-40.

29. Sebborn, M. Customising of a two-dimensional CAD system to service the needs of a small high technology company. *Computer-Aided Engineering Journal* (February, 1989), 13-15.

30. Shaiken, H. The automated factory: Vision and reality. In *Computers in the Human Context*. T. Forester, Ed. Basil Blackwell, New York, 1989, 291-300.

31. Sinclair, M., Siemieniuch, C., and John, P. A user-centered approach to define high-level requirements for next-generation CAD systems for mechanical engineering. *IEEE Transactions on Engineering Management* 36, 4 (1989), 262-270.

32. Sutherland, I. SKETCHPAD: A Man-Machine Graphical Communication System. *Proceedings of AFIPS* 23, 329-346 (Detroit, May, 1963).

33. Ullman, D., and Dietterich, T. Mechanical design methodology: Implications on future developments of computer-aided design and knowledge-based systems. *Engineering with Computers* 2, 1 (1987), 21-29.

34. Ullman, D., Wood, S., and Craig, D. The importance of drawing in the mechanical design process. *Computers and Graphics* 14, 2 (1990), 263-274.

35. Warner, T. Information technology as a competitive burden. In *Computers in the Human Context*. T. Forester, Ed. Basil Blackwell, New York, 1989, 273-280.

36. Wolz, U. The impact of user modeling on text generation in task-centered settings. *Proceedings Second International Conference on User Modeling* (Honolulu, 29 March - 1 April, 1990).