Int. J. Man-Machine Studies (1991) **34**, 161-184

Twinkling lights and nested loops: distributed problem solving and spreadsheet development

BONNIE A. NARDI AND JAMES R. MILLER

Hewlett-Packard Laboratories, Human-Computer Interaction Department, 1501 Page Mill Road, Palo Alto, CA 94304 USA

(Received April 1990 and accepted in revised form August 1990)

In contrast to the common view of spreadsheets as "single-user" programs, we have found that spreadsheets offer surprisingly strong support for cooperative development of a wide variety of applications. Ethnographic interviews with spreadsheet users showed that nearly all of the spreadsheets used in the work environments studied were the result of collaborative work by people with different levels of programming and domain expertise. We describe how spreadsheet users cooperate in developing, debugging and using spreadsheets. We examine the properties of spreadsheet software that enable cooperation, arguing that: (1) the division of the spreadsheet into two distinct programming layers permits effective distribution of computational tasks across users with different levels of programming skill; and (2) the spreadsheet's strong visual format for structuring and presenting data supports sharing of domain knowledge among co-workers.

1. Introduction

People organize themselves and their work so that problems can be solved collectively (Vygotsky, 1979; Bosk, 1980; Lave, 1988; Newman, 1989; Seifert & Hutchins, 1989). We are interested in the artifacts that support and encourage this collective problem solving. A spreadsheet is a "cognitive artifact" (Norman, unpublished manuscript; Chandrasekaran, 1981; Holland & Valsiner, 1988; Norman & Hutchins, 1988) that can be understood and shared by a group of people, providing a point of cognitive contact that mediates cooperative work. In this paper we examine the shared development of spreadsheet applications. We report the results of our ethnographic study of spreadsheet use in which we found that users with different levels of programming skill and domain knowledge collaborate informally to produce spreadsheet applications. In the first part of the paper we present a descriptive, empirical report of collaborative work practices, documenting the kinds of cooperation found among spreadsheet users, and the ways in which problem solving is distributed across users with different skills and interests. In the second part of the paper we describe and analyse the characteristics of spreadsheet software that support cooperative work.

In contrast to studies of computer-supported cooperative work (CSCW) that focus on software systems specifically designed to support cooperative work within an organization (Grudin, 1988), we address how a certain class of traditional personal computer applications—spreadsheets—function as *de facto* cooperative work environments. We describe how spreadsheet users work together, even though spreadsheets lack "designed-in" technological support for cooperative work.

0020-7373/91/020161 + 24$03.00/0

We use the term "cooperative work" in the general sense of "multiple persons working together to produce a product or service" (Bannon & Schmidt, 1989). In this paper we want to draw attention to a form of cooperative computing already well established in office environments. As we will describe, spreadsheets emerge as the product of several people working together, through not in formally designated teams, task forces, or committees. On the contrary, spreadsheet work flows across different users in fluid, informal ways, and cooperation among spreadsheet users has a spontaneous, self-directed character.

Our research highlights two forms of cooperative work that are central to computer-based work and that have received little attention in the CSCW community: the sharing of programming expertise and the sharing of domain knowledge. Because of the CSCW emphasis on computer systems that enhance interpersonal communication (e.g. e-mail, remote conferencing, shared white-boards), the importance of collaboration in programming itself has been over-looked. The current interest in "empowering users" through participatory design methods (Bjerknes, Ehn & Kyng, 1987) and end user programming systems (Panko, 1988) will, we believe, begin to draw attention to collaborative programming practices of the kind we describe in this paper. The sharing of domain knowledge has been only implicitly recognized in CSCW research; studies tend to focus on communication techniques themselves, rather than on what is being communicated. In this paper we discuss the implications of the particular visual representation of the spreadsheet for communicating analyses based on numeric data.

Since 1986 about five million spreadsheet programs have been sold to personal computer users, second in number only to text editors, and far ahead of any other kind of software (Alsop, 1989). Spreadsheets deserve our interest as the only widely used end user programming environment; text editing and drawing packages are used by many, but involve no programming. With spreadsheets, even unsophisticated users can write programs in the form of formulas that establish numerical relations between data values. Users who show no particular interest in computers *per se* voluntarily write their own spreadsheet programs, motivated by interests beyond or completely unrelated to job requirements—a claim that cannot be made for any other kind of software that we know of. In large part this is because the spreadsheet's "twinkling lights"†—the automatically updating cell values—prove irresistible. Spreadsheet users experience a real sense of computational power as their modifications to data values and formulas appear instantly and visibly in the spreadsheet.

Despite the prevalence of spreadsheets in the personal computing world, spreadsheets have not been widely studied. Kay (1984), Hutchins, Hollan and Norman (1986), and Lewis and Olson (1987) enumerated some of the benefits of spreadsheets which include a concrete, visible representation of data values, immediate feedback to the user, and the ability to apply formulas to blocks of cells. There are some experimental studies of spreadsheet use that focused on small aspects of the user interface; for example, Olson and Nilsen (1987) contrasted the methods by which subjects entered formulas in two different spreadsheet products. (See also Brown & Gould, 1987; Napier, Lane, Batsell & Guadango, 1989.) In

† We are indebted to Ralph Kimball of Application Design Incorporated of Los Gatos, California for this turn of phrase.

another type of study, Doyle (1990) reported his experiences of teaching students to use Lotus 1-2-3,† though most of his observations could apply to any kind of software (e.g. inconsistencies in file naming conventions). Other researchers have used spreadsheets as a model for various kinds of programming environments (Van Emden, Ohki & Takeuchi, 1985; Piersol, 1986; Lewis & Olson, 1987; Spenke & Beilken, 1989).

Our study began with the traditional "single-user application" perspective. We were (and still are) interested in spreadsheets as computational devices, and wanted to learn more about how spreadsheets users take the basic structure of a spreadsheet and mould it into an application that addresses some specific need. In particular, we were interested in the success *non-programmers* have had in building spreadsheet applications. We saw no reason to dispute Grudin's (1988) comments that spreadsheets are "single-user applications" in which "an individual's success . . . is not likely to be affected by the backgrounds of other group members", and that "motivational and political factors" are unimportant for spreadsheet users. However, as the study progressed, we were struck by two things:

- **Spreadsheet co-development is the rule, not the exception.** In the office environments we studied, most spreadsheets come about through the efforts of more than one person. The feeling of co-development is very strong; people regularly spoke of how "we" built a spreadsheet, and were very aware of the cooperative nature of the development process.
- **Spreadsheets support the sharing of both programming and domain expertise.** Because of our focus on end-user programming, we soon noticed that one reason spreadsheet users are so productive is that they successfully enlist the help of other, more knowledgeable users in constructing their spreadsheets. In the same way, experienced co-workers share domain knowledge with less experienced colleagues, using the spreadsheet as a medium of communication.

We do not mean to suggest that spreadsheets are never developed by individual users working completely independently. But presupposing that spreadsheets are "single-user" applications, blinds us to seeing the cooperative use of spreadsheets of which we found much evidence in our study. We will describe how spreadsheet users:

(1) share programming expertise through exchanges of code;
(2) transfer domain knowledge via spreadsheet templates and the direct editing of spreadsheets;
(3) debug spreadsheets cooperatively;
(4) use spreadsheets for cooperative work in meetings and other group settings; and
(5) train each other in new spreadsheet techniques.

We will elaborate these activities via ethnographic examples from the research.

2. Methods and informants

The ideas presented in this paper are based on our ethnographic research including extensive interviewing of spreadsheet users, and analysis of some of their spread-

† Lotus and 1-2-3 are registered trademarks of Lotus Development Corporation.

sheets which we collected during the course of interviewing. We have chosen to study a small number of people in some depth to learn how they construct, debug and use spreadsheets. We are interested in the kinds of problems for which people use spreadsheets and how they themselves structure the problem solving process—topics that by their very nature cannot be studied under the controlled conditions of the laboratory. We have also examined and worked with several different spreadsheet products including VisiCalc (the original personal computer spreadsheet), Lotus 1-2-3 and Microsoft Excel.†

For the field research we interviewed and tape recorded conversations with spreadsheet users in their offices and homes.‡ Our informants were found through an informal process of referral. We told them that we were interested in software for users with little formal programming education and that we wanted to talk to people actively using spreadsheets. The interviews were conversational in style, intended to capture users' experiences in their own words. A fixed set of open-ended questions was asked of each user (see the appendix for the list of questions), though the questions were asked as they arose naturally in the context of the conversation, not necessarily in the order in which they appear in the appendix. During the interview sessions we viewed users' spreadsheets on-line, and sometimes in paper form, and discussed the uses and construction of the spreadsheets. The material in this paper is based on about 350 pages of transcribed interviews with 11 users, though we focus on a smaller subset here to provide ethnographic detail.

Informants in the study were college-educated people employed in diverse companies, from small start-ups to large corporations of several thousand employees. Informants had varying degrees of computer experience ranging from someone who had only recently learned to use a computer to professional programmers. Most were non-programmers with three to five years experience with spreadsheets. Informant names used here are fictitious. Five sets of spreadsheet users illustrate the cooperative nature of spreadsheet development:

- *Betty and Buzz* run a start-up company with eight employees. Betty is the chief financial officer of the company and Buzz a developer of the product the company produces. Betty does not have a technical background though she has acquired substantial computer knowledge on her own, largely through using spreadsheets. Buzz is a professional programmer. They use spreadsheets for their customer lists, prospective customer lists, product sales, evaluation units, tradeshow activity and accounts receivable.
- *Ray* manages a finance department for a large corporation and has a large staff. He has an engineering degree and an MBA, and some limited programming experience. He uses spreadsheets to plan budget allocations across several different departments, to track departmental expenses and headcounts, and to forecast future budgetary needs.
- *Louis,* in his seventies, is semi-retired and works as an engineering consultant about two hours a day for a large manufacturing corporation. He has been working with Lotus 1-2-3§ for about a year, and has no other computer experience of any kind (he uses Lotus as his word processor). Louis's main application is analysing test data from his engineering simulations of radar designs. He learned Lotus with the help of his son Peter, an architectural engineer.

† Microsoft and Excel are registered trademarks of Microsoft Corporation.

‡ The interviews were conducted by the first author. We use the plural "we" here for expository ease.

§ All those in our study use either Lotus 1-2-3 or Microsoft Excel.

- *Laura and Jeremy* work for a medium size high tech equipment manufacturer. Laura is an accountant, the controller of the company. She directs a staff of eight, all of whom use spreadsheets. Laura is knowledgeable about spreadsheets but has no programming experience. Jeremy, Laura's manager, is the chief financial officer of the company. He is skilled at spreadsheet macro and template development.
- *Jennifer* is an accountant in a rapidly growing telecommunications company. She works closely with the chief financial officer of the company. Jennifer has been working with speadsheets for about five years. She took a course in BASIC in college but has no other computer science education.

Segments from the interviews will be presented at some length as we feel it is most convincing to let users speak for themselves. The segments are verbatim transcriptions.

3. Cooperative development of spreadsheets

3.1. BRIDGING DIFFERENCES IN PROGRAMMING EXPERTISE

Spreadsheets support cooperative work among people with different levels of programming skill. We have found it useful to break the continuum of skill level into three groups: non-programmers, local developers and programmers. Non-programmers have little or no formal training or experience in programming. Local developers have substantial experience with some applications, and often much more willingness to read manuals. Programmers have a thorough grasp of at least one general programming language and a broad, general understanding of computing. Local developers typically serve as consultants for non-programmers in their work environments. Local developers may in turn seek assistance from programmers.

It is also important to note that the three kinds of users vary along another related dimension: *interest in computing*. In some cases non-programmers may be budding hackers, but many are simply neutral towards computers, regarding them as a means to an end rather than objects of intrinsic interest. A key to understanding non-programmers' interaction with computers is to recognize that they are not simply under-skilled programmers who need assistance learning the complexities of programming. Rather, they are not programmers at all. They are business professionals or scientists or other kinds of domain specialists whose jobs involve computational tasks. In contrast, local developers show a direct interest in computing, though their skills may be limited in comparison to programmers as a result of other demands on their time.

Betty and Buzz's work on spreadsheets for their company's finances offers a good example of cooperation among spreadsheet users with different levels of programming skill. As individuals, Betty and Buzz are quite different. Betty has a strong focus on her work as chief financial officer, and claims few programming skills. She has limited knowledge of the more sophisticated capabilities of the spreadsheet product she uses, knows little about the features of competing spreadsheets, and relies on Buzz and other more experienced users for assistance with difficult programming tasks, training, and consulting. In contrast, Buzz has a clear technical focus and strong programming skills. He is well-informed about the capabilities of the spreadsheet product in use in the company and of other competing products, and provides Betty with the technical expertise she needs.

From this perspective, then, Betty and Buzz seem to be the stereotypical end-user/developer pair, and it is easy to imagine their development of a spreadsheet to be equally stereotypical: Betty specifies what the spreadsheet should do based on her knowledge of the domain, and Buzz implements it. *This is not the case.* Their cooperative spreadsheet development departs from this scenario in two important ways:

(1) Betty constructs her basic spreadsheets *without assistance from Buzz.* She programs the parameters, data values and formulas into her models. In addition, Betty is completely responsible for the design and implementation of the user interface. She makes effective use of color, shading, fonts, outlines, and blank cells to structure and highlight the information in her spreadsheets.

(2) When Buzz helps Betty with a complex part of the spreadsheet such as graphing or a complex formula, his work is expressed in terms of Betty's original work. He adds small, more advanced pieces of code to Betty's basic spreadsheet; Betty is the main developer and he plays an adjunct role as consultant.

This is an important shift in the responsibility of system design and implementation. Non-programmers can be responsible for most of the development of a spreadsheet, implementing large applications that they would not undertake if they had to use conventional programming techniques. Non-programmers may never learn to program recursive functions and nested loops, but they can be extremely productive with spreadsheets. Because less experienced spreadsheet users become engaged and involved with their spreadsheets, they are motivated to reach out to more experienced users when they find themselves approaching the limits of their understanding of, or interest in, more sophisticated programming techniques.

Non-programming spreadsheet users benefit from the knowledge of local developers and programmers in two ways:

(1) Local developers and programmers *contribute code* to the spreadsheets of less experienced users. Their contributions may include: macros; the development of sophisticated graphs and charts; custom presentation formats, such as a new format for displaying cell values; formulas with advanced spreadsheet functions such as date-time operations; and complex formulas, such as a formula with many levels of nested conditionals.

(2) Experienced users *teach less experienced users* about advanced spreadsheet features. This teaching occurs informally, not in training classes. Often a user will see a feature in someone else's spreadsheet that they would like to have, and he or she simply asks how to use it.

As shown in the way Betty and Buzz divide up spreadsheet tasks, the problem solving needed to produce a spreadsheet is distributed across a person who knows the domain well and can build most of the model, and more sophisticated users whose advanced knowledge is used to enhance the spreadsheet model, or to help the less experienced user improve spreadsheet skills. Compare this division of labor with traditional computing which requires the services of a data processing department, or expert system development in which knowledge engineers are necessary. In these cases, the domain specialist has no role as a developer, and domain knowledge must first be filtered through a systems analyst, programmer, or knowledge engineer before it is formulated into a program.

Our interview with Ray offers another example of co-development. Ray is a local developer who makes use of programmers for some aspects of spreadsheet development. As with Betty and Buzz, the chief difference between the spreadsheet environment and traditional programming is that more experienced users develop only specific pieces of the spreadsheet program, working directly off the basic work done by the original user. For example, Ray recently commissioned a set of Lotus macros for custom menus to guide data input for the spreadsheets used by his staff. He prefers to concentrate on using spreadsheets for forecasting future trends and allocating money among the departments he serves—his real work. Ray is not interested in becoming an expert macro writer, even though he has taken an advanced Lotus 1-2-3 class where macros were covered. In the following exchange we are looking at the custom menus:

Interviewer: ... [these menus] look like they'd be pretty useful. And who developed those for you?
Ray: A programmer down in Customer Support.
Interviewer: Okay, not somebody in your group. You just sent out the work, and ...
Ray: Yeah, well, essentially, you know, I came at it conceptually, this is what I'd like to see, and they developed it. So [the programmer] made [the menus] interactive, set up the customized use.

Ray has reached the limits of his interest in programming advanced spreadsheet features himself. But he is not limited to spreadsheets without these features; he distributes the work to someone who has more interest in such things. This task distribution is similar to traditional software development in that a user provides a specification to a developer for implementation. The difference, however, is that here the user has constructed the program into which the contributed code fits. In some sense, the roles of user and "chief programmer" (Brooks, 1975) have been merged.

Spreadsheets also support cooperation between users with different programming expertise via tutoring and consulting exchanges. For example, Louis has learned almost everything he knows about Lotus 1-2-3 from his son Peter. He avoids the manual, finding it easier to be tutored by Peter. Louis's spreadsheet use, highlights an important feature of the cooperative development of spreadsheets: because the initial effort to build something really useful is relatively small, less experienced users, having had the reward of actually developing a real application, are motivated to continue to learn more, at least up to a point. Louis is starting to have Peter teach him about controlling the presentation format; for the first several months of use he concentrated only on creating basic models of parameters, data values and formulas. In general, users like Louis successfully engage other, more experienced users in the development of their spreadsheet models. They make use of problem-solving resources—i.e. more experienced users—in a very productive manner, building on their existing knowledge in a self-paced way, as they feel ready to advance.

Distributing tasks across different users and sharing programming expertise are characteristic of many programming environments—programming in Pascal or Lisp or C would almost certainly involve such collaboration. However, with spreadsheets the collaboration is specified quite differently: the end user, usually relegated to "naive user" status in traditional software development, comes center stage, appearing in the role of main developer. Spreadsheets have been successful because they give real computational power to non-programmers. Accountants and biologists and engineers who may never have taken a computer science course build useful, often complex spreadsheet applications (Arganbright, 1986). Spreadsheet users are not "naive users" or "novices"; they command knowledge of both their domain of interest and the programming techniques necessary to analyse problems in their domain. With spreadsheets, problem solving is distributed such that end users do not rely on programmers as the indispensable implementers of a set of specifications; instead end users are *assisted by* programmers who supply them with small pieces of complex code, or with training in advance features, as they build their own applications.

4. Bridging differences in domain expertise

An important aspect of cooperative work is the sharing of domain knowledge. Because spreadsheet users build their own applications, spreadsheets allow the direct transfer of domain expertise between co-workers, obviating the need to include a programmer or other outside specialist in the development cycle. Domain knowledge flows from manager to staff since managers tend to be more experienced than those they supervise, and also from staff to manager, as staff members often have specialized local knowledge needed by managers. This direct transfer of domain expertise provides efficient knowledge sharing and helps co-workers learn from one another. Instead of transferring domain expertise to a programmer or systems analyst or knowledge engineer who may never need it again, less experienced workers directly benefit from the knowledge of co-workers.

Spreadsheets mediate collaborative work by providing a physical medium in which users share domain knowledge. Spreadsheet users distribute domain expertise by directly editing each other's spreadsheets, and by sharing templates. For example, Laura works very closely with Jeremy, her manager, in developing spreadsheets. Jeremy happens to be a skilled spreadsheet user who provides macros and tutoring that Laura and her staff use. However, the more interesting distinction to be drawn here is centered around Jeremy's greater experience with their company, its manufacturing and marketing procedures, and its managerial and budgeting practices. Spreadsheets provide a foundation for thinking about different aspects of the budgeting process and for controlling budgeting activity. In the annual "Budget Estimates" spreadsheet that Laura is responsible for, many critical data values are based on assumptions about product sales, costs of production, headcounts, and other variables that must be estimated accurately for the spreadsheet to produce valid results. Through a series of direct edits to the spreadsheet, Laura and Jeremy fine-tune the structure and data values in "Budget Estimates". Laura describes this process:

Interviewer: Now when you say you and your boss work on this thing [the spreadsheet] together, what does that mean? Does he take piece A and you take piece B—how do you divide up [the work]?
Laura: How did we divide it up? It wasn't quite like that. I think more . . . not so much that we divided things up and said, "OK, you do this page and you do this section of the spreadsheet and I'll do that section," it was more . . . I did the majority of the input and first round of looking at things for reasonableness. Reasonableness means, "What does the bottom line look like?" When you look at the 12 months in the year, do you have some

funny swings that you could smooth out? Because you want it to be a little bit smoother. So what can you do for that? Or, if you do have some funny spikes or troughs, can you explain them? For example, there's one really big trade show that everybody in the industry goes to . . . So our sales that month are typically low and our expenses are high. This trade show is very, very expensive . . .
Interviewer: So there's a spike in your [expenses and a trough in sales] . . .
Laura: Yeah. So as long as you can *explain* it, then that's OK. So what my boss did was, I would do the first round of things and then I would give him the floppy or the print-outs and I'd say, "Well this looks funny to me. I don't know, is that OK, is it normal? Should we try to do something about it?" And so what he did was he took the spreadsheets and then he would just make minor adjustments.
Interviewer: Now was he adjusting formulas or data or . . . ?
Laura: Data.

. . .

Interviewer: . . . So it was a process of fine tuning the basic model that you had developed. And then you of course had to get his changes back, and look at them and understand them.
Laura: Yes. And one thing he did do, was, he added another section to the model, just another higher level of analysis where he compared it to our estimate for this year. He basically just created another page in the model—he added that on.

In preparing a budget that involves guesswork about critical variables, Laura is able to benefit from her manager's experience. They communicate via the spreadsheet as he literally takes her spreadsheet and makes changes directly to the model. She has laid the groundwork, provided the first line of defense in the "reasonableness" checking; Jeremy then adjusts values to conform to his more experienced view of what a good estimate looks like. Jeremy also made a major structural change to the spreadsheet, adding another level of analysis that he felt would provide a useful comparison. The spreadsheet was cooperatively constructed, though not in a simple division of tasks; instead the model emerged in successive approximations as Laura and Jeremy passed it back and forth for incremental refinement.

Spreadsheet users often exchange templates as a way of distributing domain expertise. Jeremy, for example, prepares budget templates used by Laura and her staff. They contain formulas and a basic structure for data that he works out because of his greater knowledge of the business. Laura and her staff fill in the templates according to their knowledge of their individual areas. Laura and her staff are doing more than "data entry"; as in the "Budget Estimates" spreadsheet, estimates requiring an understanding of many factors often make up a significant aspect of a spreadsheet, and deriving these estimates demands thought. Users such as Laura may also specialize a template if their particular area requires additional information, such as another budget line item. The use of templates takes advantage of domain expertise at local levels, such as that of Laura and her staff, and higher levels, such as Jeremy's.

Ray's work with spreadsheets provides another example of how users share spreadsheet templates. Ray prepared "targeting templates" for his staff in order to standardize the process of targeting expenses. Because of his wider perspective looking across several departments, Ray is in the best position to develop a standard. The templates also contain the custom menus that facilitate data input. Each staff member builds the spreadsheet for his or her area on top of the template, insuring that minimum requirements for data collection and analysis are met, and insuring that the best possible information at the local level goes into the spreadsheets. Ray links them together. In these spreadsheets, problem solving is distributed over users who vary in both level of programming skill and domain knowledge: Ray, a local developer with domain expertise, provided the basic template; a programmer created the menus constructed of macros; and Ray's staff members, domain experts in their departments, supply data values for their respective areas.

5. Cooperative use

Many spreadsheets are destined from the start for the boardroom or the boss's desk or the auditor's file. In our study, spreadsheet users were very aware of the importance of presenting their spreadsheets to others—Laura stated, "I usually think in terms of my stuff [her spreadsheets] as being used by somebody else"—and users constructed spreadsheets with effective presentation in mind.

Spreadsheets are a common sight at meetings and in informal exchanges between co-workers—usually in paper copy or slide format. The use of paper copies and slides of spreadsheets is another means by which co-workers share domain knowledge. Some workers work with spreadsheets exclusively in hardcopy form and are not users of the software—for example executives who analyse and modify paper copies of spreadsheet models prepared by their staff members, and who present spreadsheets on slides and handouts at meetings.

In the following exchange we are discussing a budgeting spreadsheet Jennifer created for her company's chief financial officer. She condensed 43 pages of data from a mainframe application (prepared by the MIS department) into one summary page. We begin by looking at the MIS data:

Jennifer: These are the budget numbers. And then it shows the detail of what was purchased against those budget numbers, and when and how much month-to-date and year-to-date against those. And it shows the actual [amount] spent and variance from the budget.
Interviewer: And this really does have a lot of detail—it's down to the fabric on the chains.
Jennifer: Uh huh, ha! . . . everybody wants to know what we spent our money on, and, "How much do we have left?"
Interviewer: Now what do you do with this information?
Jennifer: . . . we have a presentation for the Board of Directors and the CFO [Chief Financial Officer] makes, but I prepare all this information for him. I compile this. I condense it onto a spreadsheet. . . . So I summarize the larger items, say, you know, the H-P 3000 [a Hewlett-Packard computer recently purchased by her company] for example. That's one of the big items that I pull out. . . . The Board of Directors does not want to see [a lot of] detail—they just want something very summarized. . . . So now it's down from 43 pages to one page. So mine shows the year-to-date budget . . . but it's all summarized into large dollar value items within each functional area.

. . .

Interviewer: Now what happens . . . when they go into the meeting and the CFO presents it? Does he explain it to the Board of Directors, or just put it up on a slide, or, what do they do with this?
Jennifer: . . . he hands out a copy to everybody, and then he puts it up on a slide, and he goes through each of the areas where they are going to be over [-budget]. And he was also presenting this so he could get approval for next quarter's budget. . . . he was showing them the . . . Q3 [Third Quarter] forecast column, and saying, "Okay, that is how much we need to approve it." And, "Where are we going from here?" Also, "What are we anticipating?"

The spreadsheet artifact is used by the CFO to organize and stimulate discussion in the Board of Directors meeting. The structures and cell values of the spreadsheet are meaningful to the board members; for example, the CFO points to the "Q3 forecast column" and individual data values such as the number showing "how much we need to approve (the Q3 budget)." Larger issues, e.g. "Where are we going from here?" are also introduced in the context of viewing the spreadsheet in the meeting.

Later in the interview Jennifer describes how the summary spreadsheet was created. The creation of this spreadsheet is an example of cooperative development; we include it here to show how development and use flow together as users collaborate in creating a spreadsheet whose ultimate purpose is a presentation to others. The final spreadsheet presented to the Board was the result of quite a multi-media production: Jennifer created the original spreadsheet in Excel, gave a paper copy to the CFO for his input, made pencil annotations on another paper copy because the CFO's changes came back via voice mail, and finally updated the on-line spreadsheet:

Interviewer: . . . What are your little pencil scribbles on here [a paper copy of the one-page spreadsheet]?
Jennifer: Oh, this is what I gave to the CFO at first, just comparing Q2 year-to-date budget to Q2 year-to-date actuals. And he said, "Well, for the board meeting I want [some other things]". Every time you do this he wants it differently. So I can't anticipate it. I just give him what I think [he wants] and then he says, "Ah, no, well, I want to have projected Q3 and projected Q4, and then total projected, and then have the whole year's plan on there". So that is what I was scribbling on here.
Interviewer: Was this in a meeting with him where he was telling you?
Jennifer: Actually he sent me a voice mail message. So that is why I take notes and go back and listen to the message again and say, "Now did I write this down right?"

Laura also described the use of spreadsheets in meetings. Her comments show that the spreadsheet organizes discussion, as we have seen in the preceding example. She notes the clarifications required to reveal assumptions underlying the spreadsheet models. Making such clarifications is often a part of meetings where spreadsheets are presented. Some spreadsheet users, including Laura, attach memos which list their assumptions (e.g. a budget allocation is based on department revenue not headcount). In the following discussion Laura describes a meeting she attended where executives are poring over spreadsheets and memos:

Laura: . . . So he [the president of the company] is sitting there and he's looking at [the spreadsheets and memos], and you're just kind of sitting there [she mimics slumping over in boredom, waiting for the president to ask a question] and he refers back and forth to various pages, whether he's looking at the budget [a spreadsheet] or whether he's looking at the last year's actuals [a spreadsheet] or he's looking at a list of assumptions.
Interviewer: So he looks at all of them?
Laura: That's right Yeah . . . And occasionally he asks a question and you say, "Oh, okay, that's this here. [She points to an imaginary spot of importance on the spreadsheet.] And you know here's this and this. And *this* was the [she waves her hand indicating a phrase like "such-and-such"] and *that* was because of [another gesture], or, "Oh, I didn't think about that!"

Laura explains how spreadsheets are used in distributed locations:

Laura: . . . And also another thing that's really classic, I mean I've experienced this before [at other jobs], is you do about as much as you can . . . and then he [the executive] gets on the

airplane to go to England [or wherever] and he's on a plane for 10 or 12 hours and he looks at [the spreadsheets] again. And he's totally uninterrupted. ... And he probably has more space up there than he does in his office! And then . . . they'll get where they are [going] and either phone call or fax.
Interviewer: To ask you a question?
Laura: Yeah. To get an explanation, or more detail, or "What did you say here? What did you assume there?"
. . .
Laura: . . . [Last year my boss and I spent a lot of time on a large spreadsheet that had to be faxed.] . . . We had to make some modifications in the spreadsheet . . . to add more types of expenses, or break things out into more detail. And we sat there together sort of hunched around the screen. We had to fax about 40 [pages of print-out]. No, it was more than that. We faxed a hundred pages to England one night . . . because they had to have it. They needed to have it prior to the meetings.
Interviewer: Wow.
Laura: So they would have an opportunity to digest it and come up with their list of questions.

As the descriptions show, though the spreadsheet provides a great deal of useful data, and is meaningful to the executives and others who use them, it does not fully expose all the assumptions in a model. However, the necessary verbal explanations are quickly produced (as in the faxed spreadsheet followed up by phone calls) because the spreadsheet developers are also the domain experts—there is no need to involve programmers or MIS personnel. While spreadsheets could benefit by better facilities for exposing assumptions, the spreadsheet artifact works as well as it does because users themselves control the process of putting information into spreadsheet models. Problem solving is handled locally, without requiring the intervention of personnel from other work groups—especially valuable, as Laura described, in fighting last minute fires.

A rather emblematic example of the cooperative use of spreadsheets is provided by Louis's meticulous black binder of spreadsheet print-outs that he carries between home (where his computer is) and office (where he has meetings). Although Louis's current spreadsheets contain none of the advanced presentation features provided by spreadsheet products (because he is just learning them), the simple print-outs are a regular feature of Louis's meetings with his colleagues as they discuss new designs for radar. It is a major benefit for Louis, an unsophisticated spreadsheet user, that the development environment and the presentation environment are the same in spreadsheets; once Louis has programmed his model, he has also created an effective presentation for group discussions, with no additional work.

Though users developing spreadsheets sometimes viewed each other's spreadsheets on-line, we found no extended examples of cooperative use of on-line spreadsheets, e.g. for the duration of a meeting. Hardcopies were virtually always used, and seemed to work well since the contents of the spreadsheet were being studied not manipulated. Productive uses of on-line spreadsheets are easily imagined, e.g. organizing a meeting around trying out different what-if scenarios and projecting the spreadsheet views overhead.

6. Cooperative debugging

In an experiment, Brown and Gould (1987) found that almost half of all spreadsheets constructed by experienced spreadsheet users contained errors. Most errors were in the formulas. Formula errors were most commonly caused by inserting erroneous cell references into formulas (pointing to the wrong cell or typing the wrong cell reference); incorrectly copying a formula so that the new formula got erroneous cell references; and putting the wrong item in a column. It is difficult to know how representative these specific types of errors are because the data consisted of only 11 formula errors, out of a total of 17 errors across the nine subjects in the study (each subject committed at least one error in at least one of the three spreadsheets they constructed for the study). It does seem likely that formula errors are more common than data entry errors since much more can go wrong in a formula.

While Brown and Gould's finding seems generally valid, if it were taken out of context—that is, out of the context of the experimenter's laboratory—it could be misinterpreted to suggest that spreadsheets in actual use are full of errors. In our study we found that users devote considerable effort to debugging their spreadsheet models—they are very self-conscious about the probability of error and routinely track down errors before they can do any real harm. Spreadsheet users specifically look for those errors that could have serious consequences. For example, a spreadsheet model with a value for department headcount that is off by one would probably have some budgetary or political implications, whereas being off by one in a forecast of annual budget dollars would not.

Debugging is a task that is distributed across the group—in particular, managers monitor their staffs' spreadsheets. Cooperation is valuable in error correction tasks (in many settings) as errors that become, through over-familiarity, invisible to their authors, are readily apparent when subject to the fresh scrutiny of new viewers.

In the following exchange we are discussing sources of error in the spreadsheets prepared by Ray's staff. Ray checks these spreadsheets himself. He uses "reasonableness checks" (inspecting values to see that they fall within reasonable ranges); footing and cross-footing;† spot checking values with a calculator; and examining formulas, recording the results of the formula checking with pencil and paper:

Interviewer: [Are the staff errors] usually in the data entry of the formulas, or does it vary?
Ray: It's mostly in the formulas. Because I think everybody is careful about making sure they have tie numbers‡ so that you can get the data in. I'm not saying it doesn't happen in data entry, but I think usually it's the formulas that are suspect. Either it's a question of the right kind of formula, or it's a situation where they weren't really careful in terms of . . . what comes first, and link it to what, and that sort of thing, they've got to be careful in that.
Interviewer: [It sounds like] you guys are pretty careful about checking things.
Ray: Yeah, we're pretty careful. Where I think it can get a little difficult is when you have a really large spreadsheet—it's a big model or something—and sometimes it's difficult to check, you know, a pretty extensive spreadsheet.
Interviewer: You mean because of the volume of data, or volume of formulas? What is it about the size that makes it harder?
Ray: You got a tremendous amount of formulas in there that are pointing all kinds of different directions, and you know, it's a pretty big pass to kind of walk back through the whole thing. So you have to be very careful.

† Making sure that the sum of row totals matches the sum of column totals.

‡ A tie number is a known quantity; it provides a sort of anchor within the spreadsheet. If a tie number is incorrect, dependent values are sure to be wrong (unless, by rare chance, incorrect values cancel each other out).

Here Ray noted the difficulty of tracing relations through large spreadsheets ("formulas that are pointing all kinds of different directions"). He finds that while his analysts are generally careful, there is room for error, so he does some checking.

Other informants described similar procedures for catching errors. Laura, for example, described how she verifies cell references in formulas by writing them down and tracing them to their origin in the spreadsheet. Like Ray, she noted that a major source of errors in spreadsheets is complex formulas in large spreadsheets.

Norman (1987) and Seifert and Hutchins (1989) argue that error in the real world is inevitable. Seifert and Hutchins (1989) studied cooperative error correction on board large ships, finding that virtually all navigational errors were "detected and corrected within the navigation team." The errors in spreadsheets could be at least a little less "inevitable" with improvements to spreadsheet software such as views showing cell relations more clearly (perhaps through the use of color, highlighting and filtering), and mechanisms to constrain cell values to allow range and bounds checking. Even with improvements, however, there would still be need for vigilance to eliminate errors, which are, as Norman, Seifert and Hutchins point out, inevitable in the real world. For spreadsheet debugging, as for tasks in other rather different domains (such as navigating large ships), a key part of the error correction solution lies in distributing the work across a group.

7. How spreadsheets support cooperative work practices

We have documented in some detail how spreadsheet users develop, debug and use spreadsheets cooperatively. We now examine the spreadsheet itself, focusing on the support for cooperative work implicit in its design. Though spreadsheets were not deliberately designed to support cooperative work, they nevertheless have two key characteristics that enable collaboration:

(1) Spreadsheet functionality is divided into two distinct programming layers—a *fundamental layer* and an *advanced layer*—that provide a basis for cooperative programming. By cleanly separating basic development tasks from more advanced functionality, the spreadsheet permits a distribution of tasks in which end users accomplish the basic implementation of a spreadsheet model, and those with more sophisticated programming knowledge provide smaller, more advanced contributions in the form of code and training. The notion of "layers" is intended to capture the different aspects of spreadsheet functionality as they relate to the user's tasks of *learning and using spreadsheets.*†

(2) The visual clarity of the spreadsheet *table* exposes the structure and intent of users' models, encouraging the sharing of domain knowledge across users with different levels and kinds of domain knowledge.

7.1. THE SPREADSHEET'S PROGRAMMING LAYERS

How do spreadsheets both meet the needs of the non-programmer and allow for the development of sophisticated applications? The answer lies in the articulation of the two programming layers: the fundamental layer, sufficient for constructing basic

† The layers do not map onto any aspect of the implementation of a spreadsheet product, or a manufacturer's description of a product.

programs, is completely self-contained and independent from the advanced layer of more sophisticated features.

The fundamental layer allows users to build basic spreadsheet models that solve real problems in their domain of interest. Users who know nothing about the advanced layer can create spreadsheets. In our study, Louis was such a user; his work was accomplished entirely within the fundamental layer, and he was just beginning to explore the advanced layer. Once users have grasped the fundamental layer, they learn the advanced layer. The advanced layer is composed of a variety of individual features that can be learned and used separately. Progress in learning advanced spreadsheet features may be very fast or very slow, depending on the user.

Because the features of the fundamental and advanced layers are independent and separately manipulated, the end user can proceed with the main programming of a spreadsheet, leaving more advanced development to local developers or programmers, or learning advanced features when they are needed. We have seen how Ray drew the line at writing macros for data entry, assigning the task to a programmer.

We now look in more detail at the fundamental and advanced layers.

7.1.1. The fundamental layer

To solve a problem with a spreadsheet, the user requires facilities for *computation*, *presentation* and *modeling*. The fundamental layer meets these needs. It is composed of two parts: the *formula language*† which enables computation; and the spreadsheet *table* which provides both a means of structuring data into a model, and a presentation format.

The formula language allows users to compute values in their models by expressing relations among cell values. Each cell value may be a constant or a derived value. A formula is associated with the individual cell whose value it computes. The formula language offers a basic set of arithmetic, financial, statistical and logical functions. To use the formula language, the user must master only two concepts: cells as variables, and functions as relations between variables. The simple algebraic syntax of the formula language is easy to write and understand.

In our study we found that most users normally use fewer than 10 functions in their formulas. Users employ those functions pertinent to their domain (e.g. financial analysis) and do not have need for other functions. Spreadsheet users are productive with a small number of functions because the functions provide *high-level, task-specific operations that do not have to be built up from lower level primitives.* For example, a common spreadsheet operation is to sum the values of a range of cells within a column. The user writes a simple formula that specifies the sum operation and the cells that contain the values to be summed. The cell range is specified compactly by its first and last cell; e.g. SUM(C1:C8) sums cells 1–8 in Column C. In a general programming language, computing this sum would require at least writing a loop iterating through elements of an array, and creating variable names for the loop counter and summation variable. Spreadsheet functions obviate the need to create variable names (cells are named by their position in the grid), and

† We refer to "the formula language" because most spreadsheet products have nearly identical languages which differ only in small syntactic details.

the need to create intermediate variables to hold results—non task-related actions that many users find confusing and tiresome (Lewis & Olson, 1987).

Once the user has created some variables and established their relations in formulas, the spreadsheet takes care of the rest. It is responsible for automatically updating dependent values as independent values change. There is no programming effort necessary on the part of the user to make this happen. The spreadsheet user's task is to write a series of small formulas, each associated with an individual cell, rather than the more difficult task of specifying the full control loop of a program as a set of procedures.

The spreadsheet table solves the presentation problems of the basic spreadsheet application. The cells of the table are used to present data values, labels and annotations. In the process of developing the spreadsheet, i.e. entering the data, labels and annotations into the table, the user is at the same time creating the user interface, at no additional development cost. Even a very simple table with no use of color or shading or variable fonts for cell entries is an effective visual format for data presentation (Jarvenpaa & Dickson, 1988; Cameron, 1989; Hoadley, 1990; Nardi & Miller, 1990.

Spreadsheet users must be able to represent the structure of the problem they are trying to solve. The spreadsheet table is a structuring device: the main parameters of a problem are organized into the rows and columns of the spreadsheet, and constants and calculated values are placed in cells. Rows and columns are used to represent the main parameters of a problem. Users know that related things go in rows and columns, and spreadsheet applications take advantage of the simple but powerful semantics provided by the row/column convention. Each cell represents and displays one variable. For calculated values, the spreadsheet associates a visual object, the cell itself, with a small program, the formula. Program code is this distributed over a visual grid, providing a system of compact, comprehensible, easily located program modules (Nardi & Miller, 1990).

What distinguishes the fundamental layer of spreadsheets from the operations a beginner user might learn in a general programming language? First, the high-level facilities for computation, modeling and presentation that we have described shield users from the necessity of working with lower level programming primitives. Users can concentrate more fully on understanding and solving their problems, with much less cognitive overhead devoted to the distraction of coping with the mechanics of the software itself.

Second, because the spreadsheet has so much "built-in" functionality (automatic update, the table as a presentation device), and a high-level language (the formula language), it takes only a few hours for non-programmers to learn to build simple spreadsheet models that solve a real problem in their domain of interest. After a small investment of time, the beginning spreadsheet user has a functioning program of real use (not a toy program or completed exercise), and also an effective visual representation of the application. The spreadsheet user's first efforts yield a complete application, rather than the partial solution that would result from writing the same application in a general programming language. The fast, early success spreadsheet users' experience motivates them to continue to use the software (Nardi & Miller, 1990; Warner, 1990; also Brock, personal communication; Flystra, personal communication).

7.1.2. The advanced layer

The advanced layer of the spreadsheet provides functionality that is unnecessary for constructing a basic spreadsheet model. We call its features "advanced" because basic work proceeds without them, not because they are necessarily difficult to learn.

The features of the advanced layer are inessential for basic work, but very useful. They are: conditional and iterative control constructs; macros; advanced functions such as database, date-time, and error trapping functions; graphs and charts; and a user interface toolkit. Each part may be learned and used completely independently of any other part. Some of the advanced capabilities are very easy to learn, such as how to change column width (the first thing Louis was learning), and others are more difficult, such as the use of macros (well-understood by Buzz and Jeremy, used in simple form by Jennifer, understood but avoided by Ray, and not known by Louis, Betty and Laura).

Users learn selected parts of the advanced layer as they need them, and as they feel ready to. Some users in our study could build a spreadsheet and significantly modify the user interface after a day-long training class, and others did nothing but build basic spreadsheets using only the formula language and modeling capabilities of the spreadsheet for several months before learning anything else. Most users do not know all the aspects of the advanced layer.†

The control constructs in the advanced layer of the spreadsheet are simple but useful. They allow users to write IF-THEN-ELSE statements within an individual formula, and to iterate functions over a cell range (a rectangular group of contiguous cells).

The user interface toolkit gives users control over column width, row height, fonts, shading, outlining, color and formatting of cell values (though not all spreadsheet products provide all of these capabilities). Spreadsheets allow users to split the screen so that non-contiguous portions of a spreadsheet may be viewed at once. The graphing and charting capabilities provide graphic views of the individual data values in the cells of the spreadsheet table. Macros allow users to reuse sequences of keystokes. "Advanced" macros provide more general facilities for data and file manipulation, screen control and controlling interaction with the user during macro execution (e.g. in Lotus 1-2-3, the macro command "GETLABEL" displays a prompt in a control panel, waits for the response to the prompt, and enters the response as a label in a cell). The advanced macros are much like traditional programming functions, but they are stored, loaded, edited and manipulated like other spreadsheet macros.

As we have noted, advanced spreadsheet features often find their way into the spreadsheets of non-programmers as code written by more skilled users. Many users reach the limits of their interest in learning advanced features, at least certain ones such as macros, and do not learn to use them. But spreadsheets also provide a growth path for those interested in continuing to learn. Because the individual features of the advanced layer are independent of one another, users can selectively

† We had the fun of stumping Buzz, during an interview, with our knowledge of the IRR—internal rate of return—function in Excel.

learn them when they wish to. Very slow progress in learning features of the advanced layer does not impede the user's ability to do constructive work. Spreadsheets provide a self-paced course of study because the features of the advanced layer are inessential for basic application development and independent of other functionality.

Over time, the distribution of problem solving tasks of an individual user changes; users take on new development tasks as they acquire knowledge of additional spreadsheet functionality. For example, Jeremy described how he "discovered" macro programming. Jeremy is an executive—the chief financial officer of his company—and has never taken a computer science class. He received his MBA from Harvard Business School just prior to the time when quantitative methods (including mandatory instruction in the use of spreadsheets) were introduced into the curriculum. In the following discussion Jeremy explains how he learned about macros from reading the Lotus 1-2-3 manual and talking to programmers. We are examining one of his macros that selects files for printing and sets up printing parameters. The macro utilizes a counter, branching, and binary variables that can have 0 or 1 as values. We have been looking at each line of the macro in detail:

Jeremy: . . . And then [the macro] compares [this variable] with the counter over here.
Interviewer: So this is real programming, basically.
Jeremy: Yes, right! And unfortunately that's what I had to do for me to be able to do this. It is exactly—a program. . . . I found that out later. I didn't realize [that I was programming]. I thought I was being very clever—I was inventing something new!
Interviewer: How did you find out later? Talking to other programmers?
Jeremy: Yeah, well, exactly. I was talking to our programmer, he came over and I showed him, "Look what I've done!" And he looked at me and he says, "Well, any time you want to be a rookie programmer on my staff, you just passed, you just made the grade".
Interviewer: But you were actually able to figure out how to do this by looking at examples in the manual?
Jeremy: That's right. Yeah, because I just mapped out: What is it that I want to do? . . . What I would like to do is to have a series of instructions and have the macro search for those instructions, and based on certain yes/no conditions either perform the operation or go to the next step. That's really all I'm after. And so I kept on looking for [branching mechanisms], and once I found them in the book I found so many different places where I could use them.

There is a gradual tendency for end users to include more complex features in their spreadsheets and to utilize local developers and programmers less. It should be remembered however, that this process may be very gradual, and would not happen at all for many users if they did not have an easy route of entry through the fundamental layer. In contrast, many students resist the frustration and tedium of learning general programming languages and do not become adept at programming in them.

In our study, spreadsheet users most commonly learnt new spreadsheet functionality in collaboration with other users. The non-programmers were extremely resistant to reading manuals (in contrast to local developers like Jeremy who kept searching the manual till he found what he wanted). Non-programmers commented that the manuals often did not explain everything they needed to know to actually use the feature they were trying to learn about. Since this meant that they would have to ask someone to supply the missing information anyway, it seemed easier to ask at the outset.†

Several users in the study, even after learning many aspects of the advanced layer, still relied on more experienced users to show them how to do new things.

7.2. THE SPREADSHEET TABLE

The strong visual representation of an application embedded in the spreadsheet table allows users to directly share domain knowledge through templates and direct edits to the spreadsheets of others, and to collectively use spreadsheets in meetings and other exchanges. Users are able to understand and interpret each other's models with relative ease because the tabular format of the spreadsheet presents such a clear depiction of the parameters and data values in spreadsheet applications.

Spreadsheets have done well at data display by borrowing a commonly used display format—that of the table. Cameron (1989) pointed out that tables have been in use for 5000 years. Inventory tables, multiplication tables and tables of reciprocal values have been found by archaeologists excavating Middle Eastern cultures. Ptolemy, Copernicus, Kepler, Euler, and Gauss used tables. Modern times brought us VisiCalc, in tabular format. VisiCalc was modeled directly on the tabular grid of accountants' columnar paper which contains numbered rows and columns. Today's spreadsheets, while much enhanced in functionality, have not changed the basic VisiCalc format in the smallest detail. A tabular grid in which rows are labeled with numbers and columns are labeled with letters characterizes all commercially available spreadsheets.

Tables excel at showing a large amount of data in a small space and in helping users to identify individual data values (Jarvenpaa & Dickson, 1988; Cameron, 1989)—precisely what is needed for spreadsheet applications because they contain many numeric values, each of which may be important to understanding an application. The perceptual reasons that tables so effectively display discrete data items are not well understood. Cleveland suggests that the notion of "clustering"—the ability to hold a collection of objects in short-term memory and carry out further visual and mental processing—applies to many visual forms (Cleveland, Unpublished data), and it seems relevant to tables. The arrangement of data items in rows and columns appears to permit efficient clustering as users can remember the values in a row or column and then perform other cognitive tasks that involve the values.

The semantics of rows, columns and cells are agreed upon and well understood by spreadsheet users. Because tables are so commonly used to display data of many kinds, most spreadsheet users are already familiar with them. Jarvenpaa and Dickson (1988) noted that many people must be taught to correctly interpret

† Manuals may be confusing at a more fundamental level. Louis gave up completely on manual reading (getting his son Peter to tutor him instead) when he could not figure out the sense in which the word "default" was being used in his Lotus 1-2-3 documentation. (Louis had a rather old copy of the manual, and the newer Lotus 1-2-3 manuals may be less confusing.) The meaning did not jibe with what he understood "default" to mean, nor with the dictionary definition, which, puzzling over the manual, he looked up. During our interview he showed us the definition. Webster's Ninth New Collegiate Dictionary defines default as "failure to do something required by duty or law"; also failure to appear in court, to pay a debt, meet a contract, or agreement, or failure to compete in or finish an appointed contest. Louis' confusion is understandable.

plotted line graphs, but most people are already practised at understanding tables. Users readily comprehend that in a spreadsheet, rows and columns are used to represent the main parameters of a model, and each cell represents and displays one variable. In looking at the spreadsheets of co-workers, the conventions of rows, columns and cells permit users to interpret the intentions of the developer.

Spreadsheets fare less well at clearly exposing the formulas underlying the cell values in the table. As we described in our discussion of debugging, checking a formula from a co-worker's spreadsheet (or from one's own spreadsheet for that matter) involves an awkward pencil and paper procedure of tracking down and verifying cell references in the formulas.† In our study we found that users do follow the pencil and paper procedures to ensure that formulas are correct, but many users cited the necessity of doing this as their main complaint about spreadsheets.

8. Implications for computer supported cooperative work

Our research focused on a single cognitive artifact—the spreadsheet. In the course of examining its structure, following it into meetings, finding out how people use it to solve certain kinds of problems, we learned two things of broad interest to CSCW research:

(1) As users gain more control over computational resources through the use of end user programming systems, cooperative work practices should be anticipated and taken advantage of by designers of such systems. Users will inevitably vary in their skill level, and computational tasks can be distributed over users with different skills through the sharing of code and training.

(2) One of the most fundamental reasons to engage in any kind of cooperative work is to share domain knowledge. Software systems that provide a strong visual format which exposes the structure and data of users' problem-solving models will support and encourage the exchange of domain knowledge.

End user software systems must provide basic development capabilities for non-programmers—what has made spreadsheets so successful is putting computational power into the hands of domain experts. In this distribution of computing tasks, development is shifted away from programmers; they supply limited but technically advanced assistance to developer/domain experts.

The layered design of spreadsheet software seems a good model for other software systems—the ability to build complete, if simple, models with basic, easily learned functionality is the key to getting users off to a quick, rewarding start. The spreadsheet provides for distributed programming by separating the basic functionality of the fundamental layer from the useful but unnecessary features of the advanced layer. End user programming systems should take advantage of the fact that local developers and programmers can reinforce and extend the programs of non-programmers through cooperative work practices—users need not be limited by their lack of programming sophistication.

Non-programmers attain rapid proficiency with the functionality of the fundamental layer because its operations are high-level and task-specific. This implies

† Some spreadsheet products provide views of the table in which the formulas are shown instead of cell values. This has its uses, but is not sufficient for formula verification because the cell values are no longer visible, and long formulas are truncated.

that end user programming systems must develop rather domain specific languages and interaction techniques whose operations will make sense to some particular set of users. The requirement that user programming languages be task-specific, contrasts sharply with the commonly advocated proposal to empower end users by helping them acquire competence in using general programming languages (Lewis & Olson, 1987; Maulsby & Witten, 1989; Neal, 1989). In general, we feel that users should be supported at their level of interest, which for many is to perform specific computational tasks within their own specialized domain, not to become computer programmers.

Just as spreadsheets distribute problem solving tasks *across* users differently than traditional computing, there is a different temporal distribution of tasks taken on by an individual user. Some users go on, over time, to learn and use new spreadsheet features (often very slowly)—in contrast to those who completely give up on general programming languages. Once users have successfully developed their own applications, they can begin to add new software techniques to their repertoire as they are ready. Through collaborations with more experienced users, spreadsheet users progress into the advanced layer. It is precisely because users have been supported by a high-level, task-specific software system that allowed them to get their work done and to experience a sense of accomplishment that they can then make progress, if they choose, in learning more general techniques. When spreadsheet users learn macros or the use of conditional and iterative facilities or formatting tricks, they venture into the realm of general programming. Such learning may occur in glacial time from the perspective of an experienced programmer, but perhaps that is appropriate for users whose primary accomplishments lie outside the field of programming.

Spreadsheets succeed because they combine an expressive high level programming language with a powerful visual format to organize and display data. Because the spreadsheet table so clearly exposes the structure and content of spreadsheet applications, co-workers easily and directly exchange domain knowledge. The shared semantics of the table facilitate knowledge transfer between co-workers; the very structure of the rows, columns and cells of the table transmits a great deal of information.

The lesson to be learned from the tabular structure of the spreadsheet is that simple, familiar visual notations form a good backbone for many kinds of scientific, engineering and business applications. Visual notations are based on human visual abilities such as detecting linear patterns or enclosure, that people perform almost effortlessly. Many diagrammatic visual notations such as tables, graphs, plots, panels and maps have been refined over hundreds if not thousands of years (Tufte, 1983; Cameron, 1989). They are capable of showing a large quantity of data in a small space, and of representing semantic information about relations among data. Like the spreadsheet table, these visual notations are simple but expressive, compact but rich in information.

We expect to see computer-based versions of tables, graphs, plots, panels and maps evolve into more sophisticated visual/semantic mechanisms, utilizing knowledge-based representations and interactive editing and browsing techniques such as filtering and fish-eye views (Furnas, 1986; Ciccarelli & Nardi, 1988). Today, visual notations are commonly used for display purposes, but it is less common for users to be able to manipulate their components—to be able to ask about the values behind a point on a plot, for example, or to expand a region on a map to show more detail. It is even less common for these displays and their components to possess any semantic information about their relationships to other displays or components—for example, constraints between specific values, or the mapping from one notation to another.

Visual notations with well-defined semantics for expressing relations will provide useful reusable computational structures. Filling a middle ground between the expressivity of general programming languages and the particular semantics of specific applications, they represent a fairly generic set of semantic relations, applicable across a wide variety of domains. New visual notations are possible and useful as Harel (1988) has shown with his work on statecharts. Statecharts formally describe a collection of sets and the relationships between them. Although Harel's work is quite new, Bear, Coleman and Hayes (1989) have already created an interesting extension to statecharts called object charts, for use in designing object-oriented software systems. Heydon, Maimone, Tygard, Wing and Zaremski (1989) used statecharts to model a language for specifying operation system security configurations.

As we have tried to show in our discussion of cooperative work practices among spreadsheet users, spreadsheets support an informal but effective interchange of programming expertise and domain knowledge. Spreadsheets achieve the distribution of cognitive tasks across different kinds of users in a highly congenial way; sojourners of the twinkling lights mix it up with crafters of nested loops—and all with software for which no explicit design attention was given to "cooperative use".

Many thanks to Lucy Berlin, Susan Brennan, Dave Duis, Danielle Fafchamps, Martin Griss, Jeff Johnson, Nancy Kendzierski, Robin Jeffries, Jasmina Pavlin and Craig Zarmer for helpful discussions and comments on earlier drafts of this paper. Thanks also to our informants, who showed great generosity in taking the time to talk to us, and provided careful explanations of their work with spreadsheets.

References

Alsop, S. (1989). Q & A: Quindlen and Alsop: Spreadsheet users seem satisfied with what they already have. *InfoWorld*, September 11, 102–103.

Arganbright, D. (1986). Mathematical modeling with spreadsheets. *Abacus*, **3**, 18–31.

Bannon, L. & Schmidt, K. (1989). CSCW: Four characters in search of a context. *Proceedings of the First European Conference on Computer Supported Cooperative Work EC-CSCW'89*, September 13–15, Gatwick, London, pp. 358–372.

Bear, S., Coleman, D. & Hayes, F. (1989). *Introducing Objectcharts, or How to Use Statecharts in Object-oriented Design*, HPL-Report-ISC-TM-89-167. Bristol, England: Hewlett-Packard Laboratories.

Bjerknes, G., Ehn, P. & Kyng, M. (1987). *Computers and Democracy: A Scandinavian Challenge*. Brookfield, Vermont: Gower Publishing Company.

Bosk, C. (1980). Occupational rituals in patient management. *New England Journal of Medicine*, **303**, 71–76.

Brooks, F. (1975). *The Mythical Man Month: Essays on Software Engineering*. Reading, MA: Addison-Wesley.

Brown, P. & Gould, J. D. (1987). How people create spreadsheets. *ACM Transactions on Office Information Systems*, **5**, 258–272.

CHANDRASEKARAN, B. (1981). Natural and social system metaphors for distributed problem solving: Introduction to the issue. *IEEE Transactions on Systems, Man and Cybernetics*, **SMC-11**, 1-5.

CAMERON (1989). *A Cognitive Model for Tabular Editing*, OSO-CISRC Research Report, Ohio State University.

CICCARELLI, E. & NARDI, B. (1988). Browsing schematics: Query-filtered graphs with context nodes. In *Proceedings of the Second Annual Workshop on Space Operations, Automation and Robotics* (*SOAR'88*), July 20–23, Dayton, Ohio, pp. 193–204.

DOYLE, J. R. (1990). Naive users and the Lotus interface: A field study. *Behavior and Information Technology*, **9**, 81–89.

FURNAS, G. (1986). Generalized fisheye views. *Proceedings of CHI'86, Conference on Human Factors in Computing Systems*, April 13–17, Boston, pp. 16–23.

GRUDIN, J. (1988). Why CSCW applications fail: Problems in the design and evaluation of organizational interfaces. In *CSCW'88*: *Proceedings of the Conference on Computer Supported Cooperative Work*. September 26–28, 1988, Portland, Oregon, pp. 85–93.

HAREL, D. (1988). On visual formalisms. *Communications of the ACM*, **31**, 514–520.

HEYDON, A., MAIMONE, M., TYGAR, J., WING, J. & ZAREMSKI, A. (1989). Constraining pictures with pictures. In *Proceedings of IFIPS'89*, August, San Francisco, pp. 157–162.

HOADLEY, E. (1990). Investigating the effects of color. *Communications of the ACM*, **33**, 120–125.

HOLLAND, D. & VALSINER, J. (1988). Cognition, symbols and Vygotskty's developmental psychology. *Ethos*, **16**, 247–272.

HUTCHINS, E., HOLLAN, J. & NORMAN, D. (1986). Direct manipulation interfaces. In D. Norman & S. Draper, Eds. *User Centered System Design*. pp. 87–124. Hillsdale, NJ.: Erlbaum Publishers.

JARVENPANA & DICKSON (1988). Graphics and managerial decision making: Research based guidelines. *Communications of the ACM*, **31**, 764–744.

KAY, A. (1984). Computer software. *Scientific American*, **5**, 53–59.

LAVE, J. (1988). *Cognition in Practice*: *Mind, Mathematics and Culture in Everyday Life*. Cambridge: Cambridge University Press.

LEWIS, G. & OLSON, G. (1987). Can principles of cognition lower the barriers to programming? *Empirical Studies of Programmers*: *Second Workshop*. pp. 248–263. Norwood, NJ: Ablex Publishing.

MAULSBY, D. & WITTEN, I. (1989). Inducing programs in a direct-manipulation environment. In *Proceedings of CHI'89, Conference on Human Factors in Computing Systems*. April 30–May 4, 1989. Austin, Texas. pp. 57–62.

NAPIER, H., LANE, D., BATSELL, R. & GUADANGO, N. (1989). Impact of a restricted natural language interface on ease of learning and productivity. *Communications of the ACM*, **32**, 1190–1198.

NARDI, B. & MILLER, J. R. (1990). The spreadsheet interface: A basis for end user programming. In *Proceedings of Interact '90*, 27–31 August, Cambridge, UK, pp. 977–983.

NEAL, L. (1989). A system for example-based programming. In *Proceedings of CHI'89, Conference on Human Factors in Computing Systems*, April 30–May 4, Austin, Texas.

NEWMAN, D. (1989). Apprenticeship or tutorial: Models for interaction with an intelligent instructional system. *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, August 16–19, Ann Arbor, Michigan, pp. 781–788.

NORMAN, D. (1987). *The Psychology of Everyday Things*. New York: Basic Books.

NORMAN, D. & HUTCHINS, E. (1988). *Computation via Direct Manipulation*, Final Report to Office of Naval Research, Contract No. N00014-85-C-0133. University of California, San Diego.

OLSON, J. & NILSEN, E. (1987). Analysis of the cognition involved in spreadsheet software interaction. *Human–Computer Interaction*, **3**, 309–349.

PANKO, R. (1988). *End User Computing*; *Management, Applications, and Technology*. New York: John Wiley and Sons.

PIERSOL, K. (1986). Object-oriented spreadsheets: The analytic spreadsheet package. In *Proceedings of OOPSLA'86*, September, pp. 385–390.

Seifert, C. & Hutchins, E. (1989). Learning from error. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society,* August 16–19, Ann Arbor, Michigan, pp. 42–49.

Spenke, M. & Beilken, C. (1989). A spreadsheet interface for logic programming. In *Proceedings of CHI'89 Conference on Human Factors in Computing Systems,* April 30–May 4, Austin, Texas, pp. 75–83.

Tufte, E. (1983). *The Visual Display of Quantitative Information.* Cheshire, CT: Graphics Press.

Van Emden, M., Ohki, M. & Takeuchi, A. (1985). *Spreadsheets with Incremental Queries as a User Interface for Logic Programming,* ICOT Technical Report TR-144.

Vygotsky, L. S. (1979). *Thought and Language.* Cambridge, MA: MIT Press.

Warner, J. (1990). Visual data analysis into the '90s. *Pixel,* **1,** 40–44.

Appendix: Spreadsheet study questions

(1) What do you do here (i.e. what are the tasks of your job)?
(2) What do you do with spreadsheets? (This question involved looking at actual spreadsheets on-line and/or in paper copy. We looked at spreadsheet structure, the use of annotations and labels, formula complexity, how spreadsheets are used during meetings, etc. as part of this question.)
(3) Who else uses this spreadsheet (i.e. of those we talk about in Question 2)?
(4) How did you create this spreadsheet (i.e. of those we talk about in Question 2)? Or alternatively, who created it and who else uses it?
(5) How accurate is your spreadsheet? How do you know?
(6) How do you find errors?
(7) How do you fix errors?
(8) Are there any problems you tried to solve with spreadsheets where the spreadsheet approach didn't work? If so, what are they and what were the problems?
(9) What is your educational background?
(10) What do you like about spreadsheets?
(11) What do you dislike about spreadsheets?
(12) What would make spreadsheets easier to use?
(13) What else would you like spreadsheets to do?