Creating Presentation Slides: A Study of User Preferences for Task-Specific versus Generic Application Software

JEFF A. JOHNSON
Sun Microsystems
and
BONNIE A. NARDI
Apple Computer

We conducted a study to investigate the use of generic versus task-specific application software by people who create and maintain presentation slides. Sixteen people were interviewed to determine how they prepare slides, what software they use to prepare and maintain slides, and how well the software they use supports various aspects of the task. The informants varied in how central slidemaking was to their jobs. The hypotheses driving the study were that: (1) some software applications are task generic, intended for use in a wide variety of tasks, while others are task specific, intended to support very specific tasks; (2) task-specific software is preferable, but is often not used because of cost, learning effort, or lack of availability; and (3) people who infrequently perform a task tend to use generic tools, while people who often perform it tend to use task-specific tools. Our findings suggest that several factors influence choice of slidemaking software, including desired quality, production time, user skill, willingness to use multiple tools, whether people work alone or in teams, and company policy. Furthermore, the task specificity/genericness of an application program is not a simple matter of *degree*, because it depends on several fairly independent software design issues. We (1) conclude that developing application software that supports all aspects of a task well is extremely difficult and (2) suggest an alternative approach that may be more fruitful: providing collections of interoperable tools and services.

Categories and Subject Descriptors: H.1.2 [**Information Systems**]: User/Machine Systems—*human factors;* H.4.0 [**Information Systems Applications**]: General; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces; I.3.4 [**Computer Graphics**]: Graphics Utilities—*application packages; graphics editors;* K.8.1 [**Personal Computing**]: Application Packages—*graphics*

General Terms: Human Factors

Additional Key Words and Phrases: Application software, interoperability, interview study, slide presentations, task analysis, task specific

Authors' addresses: J. A. Johnson, SunSoft, 2550 Garcia Avenue, MS UMPK 16-303, Mountain View, CA 94043; email: jeffrey.johnson@eng.sun.com; B. A. Nardi, Apple Computer, 1 Infinite Loop, Cupertino, CA 95014.

1. INTRODUCTION

Despite significant advances in user interface technology since the mid-1970s, the majority of households in the industrialized world do not yet own a personal computer [Fox 1995; SPA 1995]. One plausible reason is that most people are not interested in mastering and using computers per se, but only in performing tasks in familiar domains. People who use computers must often master unfamiliar concepts (e.g., files, directories, commands and arguments, control characters, cursors, modes, text strings, selection) that are about computation, rather than about the person's task or problem domain. Furthermore, computers—even those with graphical, WYSIWYG, menu-based user interfaces—require their users to supply the mapping between the objects and operations provided by the computer and the goals, objects, and operations of the task domain. It is, however, well established that people *cannot* easily provide this mapping: they cannot easily decompose their tasks into pieces that match the capabilities of today's computers, and they cannot easily combine the computer's capabilities so as to produce a solution to their problem [Hutchins et al. 1986; Lewis and Olson 1987]. People want to work in their own domain-specific idioms, not those of the computer [Fischer and Lemke 1988].

An antidote to the limitations of computer-centric software might be to provide highly task specific applications that allow people to work in the actual task domain rather than having to map that domain to the domain of computation. This idea underlies research efforts to prototype interactive applications that supply a high degree of task semantics [Fischer et al. 1989], as well as several projects to develop software development environments that support the development of such applications [Casner 1991; Fischer and Lemke 1988; Gould et al. 1991; Johnson et al. 1993; Olsen et al. 1992; Vlissides and Linton 1990].

The present study was conducted as part of an effort to better understand what it means for software to provide "task-specific" support for a task. In what ways do computer-based applications vary in their degree of support for tasks? Under what circumstances is a high degree of task support necessary or unnecessary? Because of our interest in fostering the development of task-specific applications in a variety of domains [Johnson et al. 1993; Nardi 1993; Nardi and Zarmer 1993; Zarmer et al. 1992], we conducted a study to gain some insight into the determinants of user preferences for task-specific versus generic software. The task domain we studied is the creation and editing of slides for visual presentations.

We began with a working hypothesis, but some background is necessary before we state it and describe the study. First, we give some examples of software applications and discuss their varying levels of task specificity. Next, we compare the support that various types of programs provide for slidemaking and then state our initial working hypothesis. Finally, we describe the study and our findings and offer some conclusions.

2. MOTIVATION FOR THE STUDY

2.1 Task-Specific vs. Generic Tools

What exactly do we mean by "task specific"? Consider the average kitchen. Most kitchens contain a large variety of tools. Some tools are designed for use in many different tasks, e.g., knives, bowls, spoons, stoves, pots. Others are designed for a relatively small set of tasks, e.g., blenders, peelers, tongs, basters, graters, cutting boards, butter knives. Still others are designed for one task only, e.g., fish scalers, cheese slicers, nutmeg grinders, apple corers, cookie cutters, coffee makers.

The functionality of a task-specific tool maps well onto the objects and actions in its target task domain. Consider two programs for maintaining checking accounts. The first program allows its users to work with objects such as *accounts, checks, dates, interest, fees,* and *amounts of money,* and actions such as *withdrawing, depositing, balancing,* and *auditing.* The second program makes its users work with objects such as *files, filenames, text fields, integers, data records,* and *tables,* and actions such as *loading buffers, opening windows, editing text fields,* and *inserting table rows.* The second program places more of a burden on its users: it requires them to figure out how actions on objects in the domain of computation map to accomplishing tasks in the domain of managing checking accounts. This is so even if the second program has a graphical user interface.

For many tasks in which computers are used, a large variety of software tools are used, ranging from extremely generic to extremely specific. Some people use calculators for preparing income tax returns; some use spreadsheets; and some use income tax programs designed for a particular year. Some companies do their accounting on calculators; some use spreadsheets; some use general accounting packages; some use accounting packages designed for a particular type of business (e.g., restaurants); and some use custom accounting software developed exclusively for them. For preparing organization charts, some people use painting programs; some use structured drawing programs; some use general tree-graph editors; and some use organization-chart editors. Similar series can be seen for producing family trees, creating schematic diagrams, managing inventories, and other tasks.

The application programs in each of these series provide successively more built-in semantic support for the task. The purpose of task-specific support is *not* to improve task *products*—indeed, as described later in this article, task-specific tools often limit what can be produced—but rather to improve task *processes.*

Our use of the term *task-specific* software corresponds closely to the term *task-integrated* software as used by Nielsen et al. [1986] in a study of software used by business professionals. Though their study focused on the use of integrated software in business, not all integrated software is *task*-integrated in their view. For example, a program containing word processing, spreadsheet, and drawing functionality, but no significant

semantic support for any particular application of that functionality, would be considered integrated but not task integrated by Nielsen et al. [1986]. We will return to this point in the Conclusions section.

Nielsen et al. [1986] also refer to specialized, standalone application programs, but mean by that something independent of task specificity. Whereas they would classify a simple text editor as "specialized" because it is not integrated with anything else, we would call it "generic" because it is nearly devoid of task semantics. On the other hand, a standalone family tree editor that embodied semantics of how family trees are constructed, which we would consider "task specific," would also be considered by Nielsen as "specialized," again because of its lack of integration. Thus, whether a program is "specialized" in their terminology has nothing to do with whether it is "task specific" in ours.

After we have described our interview study, we will have more to say about application software task specificity.

2.2 Creating Presentation Slides

Slide preparation is a task for which a wide variety of computer-based tools are used. By *slides,* we mean both 35mm slides and overhead transparencies. People prepare slides using text editors, desktop publishing systems, painting programs, drawing programs, spreadsheets, statistical analysis programs, business graphics programs, animation programs, and, recently, presentation-making programs.

Many people use structured drawing programs for making presentation slides. With drawing programs, users place manipulable graphical objects on a canvas. Text and graphics, once placed, can be edited, moved, or copied. However, the drawing programs' degree of support for slidemaking is limited. They offer, for example, no notion of a presentation *set* of slides. Users can compensate by putting slides into separate files and grouping them in folders or directories, or by placing all the drawings of a presentation together on one canvas; but such workarounds are inconvenient and inefficient. Additional drawbacks of drawing programs as tools for slidemaking are:

— They provide no support for consistency of format, font, and layout in a presentation. To have the same margins on every slide, drawing program users must painstakingly arrange things that way, separately for each slide. If the formatting requirements change, users must change every slide.
— Standard slide content, such as logos, headers, and footers, must be explicitly placed on every slide, and changes require editing every slide.
— They provide little help in changing the structure of a presentation's content. Splitting one slide into two requires much explicit copying, moving, and deleting of graphic objects.
— They provide no support for the actual presentation, e.g., attaching speakers' notes to a slide.

Put succinctly, drawing programs lack the concepts of *slides, relationships* between slides, and *presentations*. Most of the other types of software used for preparing slides also lack support for the process of creating and editing presentation slides. Nonetheless, they are commonly used for that task.

In recent years, software designed specifically for creating presentations has become available (e.g., Charisma™, Persuasion™, and PowerPoint™). Since such programs are intended only for slidemaking, they can provide much more support for that task. Slides comprising a presentation are contained in one file. The format and common content of slides are specified once for the presentation, rather than separately for each slide. Typical slide-editing actions, like removing or adding a level of detail, are explicitly supported. Finally, many such programs support the task of giving the presentation.

Note that—given enough time, skill, and talent—any slide or presentation that can be produced with a presentation program can also be produced with a drawing program. Presentation programs facilitate the *process* of creating, editing, and maintaining presentations by providing built-in domain knowledge. Supporting the process of performing a task may be more important for the usability of a software application than getting the user interface right—at least as the term "user interface" has conventionally been used. We illustrate this with a story from the experience of the first author.

In the early 1980s, at a company where the first author worked, employees made presentation slides using a text editor in conjunction with a slide-formatting program. To make a set of slides, employees created a text file containing the textual slide content and embedded formatting commands. The text file was then "compiled" using the slide formatter, producing a set of graphics files containing images of the slides. Employees had many complaints about this process. The formatting commands were hard to learn; it was hard to tell how a particular slide would look from its source file; one had to "compile" the entire set to check how a single slide looked.

When interactive painting and drawing programs became available, most employees switched to them immediately. The new programs were easier to learn and provided much better feedback than did the text editor/formatter combination.

However, people soon learned that the new programs, for all their user friendliness, did not support slidemaking very well. With the new programs, it was hard to obtain consistent formatting, hard to manage sets of slides, and hard to edit slides. After a short honeymoon, most employees switched back to the old programs, occasionally using the new ones to enhance slides generated by the slide formatter.

At the time, this mass retreat to the old tools was perplexing. People had, unaccountably, eschewed the advantages of direct-manipulation interfaces and WYSIWYG interaction to return to an old-fashioned text-based command language and tedious edit-compile-debug cycle.

2.3 Working Hypothesis

As we began our research on user-centered development of task-specific applications [Johnson et al. 1993], we explained the foregoing story as follows. People abandoned the new tools because these tools were not *task specific*. They were not *slidemaking* programs, but rather generic painting and drawing programs. Though the new tools made some aspects of the slidemaking task easier, they made others harder. The aspects of the task that became easier were those in the domain of the "user interface"; they had more to do with controlling—and learning to control—the software than with making slides. The aspects of slidemaking that the new tools made more difficult were the deeper, more task-related aspects, e.g., assuring graphical consistency between slides. While employees could, with effort and talent, make nicer-looking presentations with the new tools, more people could produce *acceptable* presentations much more quickly with the old tools. Having a "task-friendly" application was more important to users than was having a "user-friendly" interface.

We also reasoned that the *order* in which people learned these tools played a role in their ultimate preference. After switching to the WYSIWIG tools, they were willing to switch back to the textually controlled slide compiler despite its being hard to learn, because they did not have to learn it; they *already knew* how to use it. Had the slide compiler become available *after* the WYSIWIG-but-generic drawing tools, most employees—mainly those who produce relatively few slide presentations—would probably have stayed with the generic drawing and painting tools. The effort required to learn to use the slide compiler would not have been worth the expected return. We assumed that people who produced slide presentations frequently, but learned the generic painting and drawing tools first, would probably invest the time to take advantage of the slide compiler's greater semantic support for slidemaking.

Stated more formally, our initial analysis and experience with interactive computer-based applications led us to the following four-part hypothesis:

(1) *Task specificity:* The more knowledge of a particular task is built into a software application, the more specific it is to that task, i.e., the less applicable it is to other tasks. Software applications can be considered to occupy a position along a continuum, from completely generic to completely task specific.

(2) *Task-specific tools preferred:* It is best to have a tool designed specifically for the task one is performing. For example, for preparing schematic drawings, a schematic editor is preferable to a drawing editor. Good support for a task is even more important for overall usability and productivity than is a good "user interface" in the traditional sense of the term.

(3) *Limited market:* The more task specific a software application is, the smaller its potential market, requiring the developer to either charge a higher price or be satisfied with less revenue.

(4) *Factors of choice:* People use generic applications when ones that are more task specific are either unavailable, cost too much, or require too much (incremental) learning effort. In today's software market, and given the state of application development technology [Myers 1989; Zarmer and Johnson 1990], these conditions usually hold. Most computer users therefore "get by" with generic tools—e.g., text editors and drawing editors—because their level of need does not justify the cost of obtaining and learning to use task-specific ones. People prefer task-specific tools when they perform a task frequently. Their level of use justifies the overhead of acquiring and learning to use the tool.

Based on these beliefs, we (and other colleagues) began developing an Application Construction Environment (ACE) designed to facilitate the development of task-specific software applications [Johnson et al. 1993; Nardi and Zarmer 1993; Zarmer et al. 1992]. One goal of ACE was to make development of many business applications easy and cheap enough that it would be cost effective to develop highly task specific applications for small, specialized markets and short-term tasks. This was done by prepackaging commonly needed functionality in extensible application frameworks that could, like spreadsheets, be specialized for specific tasks [Nardi 1993; Nardi and Miller 1990; 1991]. A related goal was to promote a development process based heavily on task analysis and to move the center of the development process much closer to the computer users, who best understand their goals and tasks. A third goal was to provide support for representing task semantics more explicitly than do conventional software development tools.

As we developed ACE, we were aware that our four-part hypothesis should be empirically tested. Over time, we began to suspect that parts of it were incorrect. For example, a colleague who had previously worked as a graphic artist producing presentation slides for others indicated that experienced professional slidemakers prefer generic drawing and painting software for creating slides because it does not restrict them from doing what they want, while dedicated slidemaking programs often impose oversimplified views of the task and restrict the results that can be produced. Based on this counterclaim and on our own further analysis of the nature of task specificity, we decided to conduct an empirical study as a first step in evaluating and correcting our working hypothesis. We chose the domain of slidemaking because of the large variety of software tools used for that task, the relative accessibility of informants, and, not least, the challenge posed by our graphic artist colleague.

3. METHOD

We conducted an interview study to examine how people use computer software to create presentation slides. Since the marketplace for software used in slidemaking changes rapidly, it is important to note that we conducted the interviews in January through February of 1992. As reference points, the versions of Microsoft Word™, Microsoft PowerPoint™, and

Aldus Persuasion™ in use among our informants then were 4.0, 2.0, and 2.0, respectively.

3.1 Informants

The informants were 17 people whose jobs involved creating, editing, and maintaining slide presentations. Two were a husband and wife who worked—and were interviewed—together; for the purpose of this study, they were treated as one informant. All informants were college educated with several years experience making slides. They worked for a variety of companies, ranging from one-person independent consultantships to large multinational corporations in the San Francisco Bay Area (most outside of Silicon Valley). Six of the 16 informants worked in research or marketing and made slides for their own use in presentations, with slidemaking being only one of many of their job responsibilities. The other 10 informants can be considered professional slidemakers; they had, as a significant (for some, dominant) part of their job, the creation of presentation slides for others, in a variety of business areas: legal, advertising, research, and general business.

Our informants were quite happy to talk about their slidemaking software. Several warned when scheduling the interview that their busy schedule could accommodate only a brief interview, but then in the interview they seemed willing to talk for as long as the interviewer would listen. People apparently have strong opinions, both positive and negative, about the software they use.

3.2 Procedure

We developed a set of questions that covered the issues of interest in this study (see Appendix A). Most of the interviews were conducted at the informant's workplace, often with a computer slidemaking system ready-at-hand so that we could see the user's work online.

The interviewer began each interview by explaining that the purpose of the study was to learn what is involved in making slide presentations, what sorts of software people use for the task, and what people's reasons are for using or not using various software tools. The interviewer then asked the informant to describe the entire slidemaking process, from start to finish. The interviewer allowed the conversation to flow, more or less, naturally rather than strictly following the list of questions, but made sure that answers to each of the predetermined questions were captured on tape. The interviewer did *not* explain the distinction between task-specific versus generic software, or our initial working hypothesis. Interviews ranged from 1 to 3 hours per informant.

Interviews were audiotaped, then transcribed onto computer text files. About 250 pages of transcripts resulted from the interviews.

3.3 Data Analysis

We read transcripts of each interview, in some cases referring to the audiotape to clarify transcription problems or informant intent. A summary

was made of each interview that included: the informant's job role and involvement in slidemaking, the context in which slides were being produced, a summary of the slide production process as described by the informant, the software the informant uses or has used for slidemaking, the informant's reasons for using it, software features that the informant considered useful or a hindrance in slidemaking, and informant comments (if any) that seemed especially germane to the study. From these summaries, we constructed tables summarizing our findings and computed some summary statistics.

4. RESULTS

The 16 informants in our study reported using a total of 25 software programs for preparing slide presentations. The average number of programs per informant was 4.4, ranging from 1 to 8. The average was 5.6 for our informants who were professional graphic artists, and 2.5 for our other informants. All six who were not graphic artists ranked below the median number of programs used for slidemaking, which would be highly improbable by chance. This is of course not surprising: one would expect graphics professionals to have a larger collection of tools for slidemaking than amateurs.

4.1 Tabulation of Interview Data

Table I lists all of the types of software used by informants in preparing presentations and gives examples of each software type. "Desktop publishing" refers to WYSIWYG document editors and page layout programs. "Document compiler" refers to programs that compile text containing embedded formatting commands into formatted documents. Table I shows that even though the number of informants interviewed in our study was relatively small (16), our informants, as a group, used a wide variety of types of software to prepare slide presentations.

Readers who wish to see a "raw" tabulation of the interview data should refer to Appendix B. Appendix B also includes a discussion of methodological issues we encountered in tabulating the interview data. For present purposes, tables that aggregate the raw data, thereby showing tendencies, will suffice.

Table II categorizes informants into types (professional slidemaker versus amateur) and indicates, for each type of software, what types of users used it. It separates creating or organizing presentations from creating presentation content. It shows that for creating and organizing slide presentations, presentation and desktop publishing programs were the two most popular types, for both professional and amateur slidemakers. For generating presentation content, drawing and charting programs were popular among professional graphic artist informants. Informants who were slidemaking amateurs tended to stick to one program and not use auxiliary software to produce slide content; the few who used auxiliary programs mainly used spreadsheets to produce charts.

Table I. Software Types Found in Study, with Examples of Each Type

Software Type	Example Software
Presentation	PowerPoint™, Persuasion™, Charisma™
Drawing	MacDraw™, Illustrator™, Corell Draw™
Painting	MacPaint™
Charting	CricketGraph™, Harvard Graphics™
Desktop Publishing	FrameMaker™, Word™, PageMaker™
Document Compiler	LaTeX, troff
Animation	Macromind Director™
Spreadsheet	123™, Excel™
Database	Paradox™
Image Processing	PhotoShop™
Custom	--

Table III groups the specific types of software (e.g., drawing, painting, charting) into more general categories (e.g., Graphics), and indicates what types of users used what *combinations* of these software categories. For the purposes of Table III, spreadsheet, database, and custom programs were included in the Graphics category along with drawing, painting, and charting programs because informants told us that for slidemaking they used those types of software to generate data-driven graphics. Desktop publishing programs and document compilers were categorized as Document. Table III shows that the two most common combinations of software tools used by graphics professionals were Graphics only (four professionals) and Presentation, Document, and Graphics (four professionals). It also shows that professional graphic artists tend to use a larger collection of software programs for preparing slides than do "amateur" slidemakers, i.e., people who create presentations only as a small part of their jobs. The "graphics professional" listed as using only Presentation software was in fact a group of graphics clericals, referred to as "secretaries" by the informant who managed them. Among graphics amateurs, who tend not to use more than one program for making slides, Document software was the most popular category.

5. DISCUSSION

As the tabulated data show, the main finding of the study was that our original hypothesis was right in some respects and wrong in others and that the truth is more complex than either we or our graphic artist colleague understood. Four out of six of the informants for whom slidemaking was peripheral to their job used whatever general-purpose software they used for other work, e.g., document production. However, two such informants took the time to learn how to use task-specific presentation

Table II. Number of Users (Professional and Amateur Slidemaker) Using Each Type of Software

Creating and Organizing Presentations			
Software Type	**Professional**	**Amateur**	**Total**
Presentation	8 of 10	2 of 6	10 of 16
Drawing	3 of 10	1 of 6	4 of 16
Desktop Pubs	7 of 10	3 of 6	10 of 16
Doc Compiler	0 of 10	2 of 6	2 of 16
Animation	4 of 10	0 of 6	4 of 16
Creating Presentation Content for Export into Other Programs			
Software Type	**Professional**	**Amateur**	**Total**
Painting	4 of 10	1 of 6	5 of 16
Drawing	7 of 10	1 of 6	8 of 16
Charting	5 of 10	0 of 6	5 of 16
Spreadsheet	3 of 10	3 of 6	6 of 16
Desktop Pubs	2 of 10	0 of 6	2 of 16
Image Processing	3 of 10	0 of 6	3 of 16
Database	0 of 10	1 of 6	1 of 16
Custom	1 of 10	0 of 6	1 of 16

Informants who use more than one type of software for slidemaking are counted once for each type of software they use.

programs. Though no statistical tests are feasible here, the trend favors our initial hypothesis that people who create few presentations tend to use generic software to prepare slide presentations.

However, contrary to our hypothesis but in agreement with the claims of our graphic artist colleague, few of our informants who are full-time slidemakers rely mainly on task-specific presentation programs. Most made extensive use of generic software such as drawing, painting, and word processing programs rather than sticking to slidemaking programs. In the following sections, we describe the factors affecting software choice that emerged from our interviews, then revise our analysis of task specificity.

5.1 Factors Affecting Choice of Slidemaking Software

We found that the choices of software to accomplish a particular slidemaking task are highly dependent on specific requirements. Furthermore, the requirements are not fixed for a given person; they vary from specific task to specific task within the domain of creating and maintaining slide presentations. User requirements vary according to several factors.

5.1.1 *Presentation Quality, Production Time, and User Skill Level.* One factor that influences professional slidemakers' choice of software is the

Table III. Usage of Combinations of General Software Category by User Type

Combinations of Software						
Presentation	Document	Graphics	Animation	Graphics Professional	Graphics Amateur	Total
■				1	0	1
	■			0	2	2
		■		4	1	5
			■	0	0	0
	■	■		1	1	2
■	■			0	1	1
■		■		0	1	1
■			■	1	0	1
■	■	■		4	0	4
■		■	■	1	0	1
■	■	■	■	2	0	2

The document category includes desktop publishing and document compiler software. The graphics category includes drawing, painting, charting, image processing, spreadsheet, database, and custom software. The Graphics Professional column includes four people managed by informants as well as the informants themselves.

desired quality of a presentation. Closely related to this is the time one has or takes to produce the presentation. Our data make it clear that slidemaking tasks vary a great deal in requirements. Some presentations are for coworkers, and some are for external customers. Some presentations are considered ordinary while others are "fancy" or "very important." Most presentations are relatively mundane; others have millions of dollars riding on the impression they make. The goal of producing a fancy presentation leads to different choices than the goal of producing an ordinary presentation. Each kind of presentation entails organization, illustration, and other subtasks, but the differing goals mandate optimizing different aspects of the overall process.

Presentation slides are often produced on very short schedules, with production time being more important than illustration quality. To quote from two of our informants:

> Usually speed is an issue here rather than quality. . .it's always like down to the last minute.
>
> They always wanted everything yesterday. They will come to me with very little time to turn around slides.

Though many presentations are prepared on tight deadlines, not all are. Some—usually the very important, fancy ones—are anticipated and planned well in advance.

Many slidemaking organizations use different software, processes, and even personnel for producing plain presentations—the majority—than they

do for producing fancy or very important ones, of which there are relatively few. To produce plain presentations quickly, professional slidemakers tend to use dedicated slidemaking programs (though factors such as familiarity and availability sometimes limit this tendency). This is as we had predicted. However, for fancier presentations, professional slidemakers usually use generic drawing, painting, desktop publishing, or animation software, though sometimes with presentation programs serving as the final container and organizer.

Some firms employ less-skilled graphics personnel to produce the simpler slide presentations, and more highly trained graphic designers and artists to produce fancy graphics and presentations. For example, one firm has a specialist who creates high-quality color presentations and fancy graphics using generic illustration software, and "everyone else just does straight charts and graphs" and "word slides" using a presentation program. Our initial working hypothesis distinguished only between people for whom slidemaking is peripheral to their job (e.g., researchers who give conference papers) and those for whom slidemaking *is* their main job, but our interviews made it clear that the latter category consists of two quite different sorts of workers. Some full-time slidemakers are highly trained artists; they have a lot of talent, skill, and knowledge in the task domain (e.g., presentation style, graphic art). They tend to use tools that provide more freedom to exercise their domain knowledge and creativity, i.e., collections of generic software. Other full-time slidemakers are graphic *clericals;* they may have training and skill at operating their tools, but their training and skill in the graphic art task domain is low compared to graphic artists. These are people who are either (1) more concerned about being productive than they are about acquiring graphic art skills or (2) in jobs allowing them less autonomy and creativity. They tend to use tools that provide the bulk of the task knowledge (and assumptions about what sorts of presentations one will want) built-in, i.e., presentation-making programs. This explains why some of our findings regarding full-time slidemakers initially seemed split between those conforming to our original hypothesis and those that agreed with the claims of our graphic artist colleague.

One aspect of presentation quality is the quality of the graphic art. Highly trained graphic artists often use generic illustration software because the drawing capabilities of slidemaking programs are insufficient and limiting from their point of view. Here are some illustrative quotations from our interviews:

> Persuasion's not the best drawing tool.
>
> . . .the graphic tools in Persuasion are kind of low-end, not very powerful. Persuasion's drawing tools are too weak.
>
> [With MacDraw] you have more control.
>
> You can do it in PowerPoint, but depending on the art, sometimes it's faster. . .to do it in MacDraw and paste it in.
>
> I think the main point about why we use MacDraw is because, yeah. . .Persuasion would be better for a lot of word slides. . .But nobody's willing to simplify

> their graphs that much. You know? It's like they would have to. . .work at such a simple level to make a presentation, that nobody, they can't, the [clients] can't cut down on the complexity of their slides, to be able to fit in with the limitations of a program like that.
>
> Persuasion and PowerPoint are sort of integrated programs, and they're good for someone who isn't a power word processor, who isn't a power graphic artist, where they basically want to type in their own headers and dot points; and it's great for that. . . . But if you have to go beyond that where you're. . .doing real serious word processing, or doing some real elaborate graphics, it just doesn't cut it either way. . . . *There hasn't been any software that does everything well.* [emphasis added]

Because of this, professional graphic artists who create illustrations for slides tend to use generic drawing and painting tools, which give them the freedom to use their skills and to produce the illustrations they want. Some illustration programs provide image-enhancing features such as antialiasing, three-dimensional effects, and highlighting, which are absent in dedicated slidemaking programs. This is similar to the findings of Nielsen et al. [1986], although as explained in the Introduction, their terminology differs from ours:

> As the example with presentation graphs shows, a key reason for the lack of use of integrated packages seems to be *lack of functionality* in them compared with specialized stand-alone application programs. In many cases we found users who simply needed the added functionality of a specialized program and therefore used it for one of the applications they theoretically had available in their integrated package [Nielsen et al. 1986, p. 164].

Professional slidemakers producing fancy presentations provide a mirror image example to our story in which people retreated from new "user-friendly" generic illustration software to their old slide formatter. The people in that story did not want or need to optimize illustration; for them the goal was to produce acceptable slides quickly, rather than to produce beautifully illustrated slides. In contrast, a professional slidemaker preparing a fancy presentation is most concerned about graphic quality.

5.1.2 *Willingness to Use Multiple Programs in Concert.* To get the functionality they require, professional slidemakers often use slidemaking programs as only one of a set of interoperable tools. This is clear from Tables II and III. What is not shown in the tables is that professional slidemakers often use slidemaking programs, ironically, not to *make* slides but rather to *contain* and *organize* them. For example, one informant said:

> At [company X], they use PowerPoint. They offer MacDraw, Freehand, Illustrator for the illustration part of it. But it's all driven by PowerPoint. It's all put in PowerPoint.

Thus, the programs that we had regarded as task-specific tools that supported, end to end, the entire slidemaking process, were in many cases being used as tools to support a specific *subtask* of slidemaking, namely, containing and organizing slides for presentations. However, some graphic

artist informants ignored even the organizing functionality of dedicated presentation software. One informant, for example, told us she used a drawing program for that purpose:

> Everything comes through MacDraw Pro no matter where it starts.

Of course, even though there has been much progress in software integration and interoperability since Nielsen et al. [1986] conducted their study of software use, few programs used by our informants were designed to be used together, and not surprisingly they often cannot be. Interoperability is very important to professional slidemakers: they do not think much of dedicated slidemaking tools that cannot easily accept text and graphics from a variety of sources. One feature that several informants said is missing from most of their tools is the ability to *share*—rather than copy—content between applications, allowing a single piece of content to appear in multiple presentations at once and to be updated everywhere automatically when the source is changed. Macromind Director™ and, to a lesser extent, Framemaker™ were the only programs that our informants used that possessed this kind of interoperability.

5.1.3 *Teamwork.* A fourth factor in determining the users' choice of slidemaking software is whether the degree of teamwork supported or allowed by the software matches the work practice of the users' organization (see Nardi [1993] for a discussion of collaborative application development practices). Most slidemaking programs are designed to support an individual who produces slides alone. However, we found that, in many settings, people work in teams to produce and maintain slides. Existing dedicated slidemaking programs make it difficult to do this. According to one informant:

> I looked at Persuasion, because everybody was saying that Persuasion was great. And I think a bunch of the secretaries. . .used it as well. And I think the programs that are that specific are very well designed for a person who is going to sit down and think up a presentation and create the presentation right there. But the way we work is that, you know, there are dozens of people out there thinking up things, and we integrate presentations for all of them. And so for us to be able to distribute that work amongst enough people to get it done, we need to break it down into smaller units. . . . For each job here, if we used Persuasion, each job. . .would have its own. . .file with all of its slides in it. But slides get used from one job to another. . . . And so, I think [that] because of that it wouldn't work. The outlining, you know, is wonderful. But it's really designed for a different type of work atmosphere. It's is designed for the guy who's sitting down and going to do his own presentation.

Though more-generic tools do not provide real *support* for team production, they at least do not *interfere* with it in the sense that they impose little structure on the process at all. Whereas presentation programs keep an entire presentation together on one file, generic programs produce small *pieces* of presentations, allowing—indeed, requiring—users to produce, organize, and distribute the pieces as needed. While task-specific software

could, in principle, support teamwork, today's dedicated slidemaking programs do not encourage sharing of slides and files, either between individuals or between presentations. This seems to be an important factor in some people's choice to use generic software for slidemaking.

5.1.4 *Company Policy.* Some of our informants reported that the software tools they used for slidemaking were determined by company policy or simply by what software the company already owned. Most of our informants who were not professional graphic artists said that they decided themselves what software to use to produce presentations for their own use. One such informant initially used PowerPoint but switched to Persuasion because the latter is what the corporate graphic art department used and because he wanted to be able to exchange presentations and graphics with them easily.

Among informants who were professional slidemakers, corporate graphic artists apparently have more influence over the choice of software than do freelance artists. Of the six corporate graphic artists we interviewed, three claimed that they decided unilaterally what software to use, and three claimed that they were involved in the decision. Of the four freelance graphic artists interviewed, none said that they decide unilaterally; three said that they were involved in the decision; and one said that the software is determined by the client company. The last of these was actually a husband-wife team who worked as freelance graphics consultants and used different tools at each of two different client companies because each company had previously developed slidemaking procedures around those tools. Each company had decided that one program (PowerPoint™ in one; PageMaker™ in the other) would be the container/organizer for slide content created using other programs. Finally, it is clear that the software tools used by graphics clericals who produce "ordinary" business presentations are much more likely to be determined by management than are the tools of graphic artists who produce the "fancy" presentations.

Clearly, company policy plays a role in deciding what software is used. Of course, company policy is often based on history: as one informant pointed out, because presentation content and formatting are often reused, it is difficult for a company to change slidemaking software once a large archival base of presentations has been built up.

5.2 Rethinking Task Specificity

Our findings indicate to us that our initial analysis of task specificity was naive about both parts of the term "task specific."

First we had an overly simple notion of *task*. We assumed that tasks exist in some a priori sense, i.e., that there is a set of preexisting, well-defined tasks that people perform, fixed across circumstance. We regarded *creating slide presentations* as *a* task. In fact, the tasks people are trying to accomplish when they create or edit slide presentations vary tremendously. More useful concepts for capturing invariance across situations and for understanding similarities and differences between application programs

are provided by classical software engineering and activity theory. Software engineering provides the concept of a *task domain* [Prieto-Diaz 1990]: application programs are designed to support people working within a *task domain* rather than on a *task*. A task domain is defined—independent of any particular software program—in terms of what objects exist to be acted upon by the user and what actions can be performed on those objects. Activity theory provides a conceptual framework for understanding how people's goals interact with a task domain to give rise to activity [Nardi 1996]. It suggests that within a task domain, people's goals, and hence their tasks and activities, vary: different people, or the same person at different times, may need to carry out different actions within the domain and may have different criteria for success.

Second, we oversimplified what it means for a tool to be *specific* to a task. We had originally regarded task specificity as being a continuum, with completely generic tools at one extreme and ones that are totally task specific at the other. In fact, tools—including application programs—vary in several different ways that are related to the notion of task specificity:

— *Relative support for actions within a task domain:* Real-world task domains contain many objects and actions. Tasks in such domains require the execution of multiple actions, which appear in people's behavior as different activities. For example, creating a slide presentation may include creating a topic outline, writing text content, drawing graphic content, designing a consistent format, determining the slide order, producing transparencies or photographic slides as output, producing talking notes, filing and retrieving slides, and other activities. Software is expensive to develop, making it infeasible for any program to provide ideal support for all activities within the target task domain; developers must choose which ones to focus limited resources on in designing their program. For example, many dedicated slidemaking programs provide good support for creating textual content, but poor support for creating graphical content (see below). According to one graphic artist informant:

> PowerPoint is like one of these. . .software packages that try to incorporate everything, yet no particular area is very strong.

Application programs therefore, differ from one another in which *parts* of the users' task they support best.

— *Support for actions that are common across task domains*: Because of the problem of limited development resources described in the previous paragraph, some software developers maximize their market by designing software tools to support activities that are common to many task domains (e.g., creating outlines, drawing figures). The target task domain of such tools is a subdomain of many larger task domains. Application programs may therefore be *specific* in that they provide good support for a narrow activity, but *generic* in that they may be used for tasks in quite different task domains.

— *Support for coordinating activity and organizing products*: In most task domains, one distinguished activity is *coordinating and guiding the execution of actions*, i.e., making sure they are executed in the right order, each with the right inputs and tools. Another distinguished activity is *storing and organizing the products of steps* in the overall task. These activities concern the overall process, rather than any part or intermediate product of it. Again, because development resources are limited, some software tools focus on supporting these activities. For example, some slidemaking programs are good for producing an organization for a slide presentation from an outline, or for maintaining and organizing slide presentations, but poor for creating slide content. To provide explicit support for the overall process, a tool must embody significant task domain knowledge. To the extent that a tool succeeds at this, it can relieve its users of the need to have or supply it. In contrast, when a person uses a tool that supports only one of many activities in a task domain, the person must supply the domain knowledge, because the tool does not. Tools that support the overall process necessarily make assumptions about how their users will want to perform tasks in the domain and organize the products of their work. Those assumptions are crucial: they may be wrong for a given individual or organization or may be based on a naive understanding of the task domain.

— *Specializability*: Some tools are *designed* to be specializable for tasks and as such do not have fixed task specificity. For example, a power beater has different attachments, which make it into a batter mixer, a blender, a dough kneader, etc. Similarly, some software is designed to be specializable. Spreadsheets programs, most computer-assisted design (CAD) systems, some word processing programs, and other programs provide extension facilities such as formula and macro languages, stylesheets, and templates to allow users and local developers to add semantics to support specific tasks. For example, one graphic artist informant told us that at his company the preferred tool for slidemaking is a document editor, which through the use of stylesheets and template files is specialized to provide good support for creating and editing presentations, even very high quality ones.

The contrast between some of these different aspects of task specificity is exemplified in kitchen tools by a vegetable peeler and a breadmaking machine. A vegetable peeler is highly specific in the sense that it supports a narrow activity well, but generic in the sense that it can be used in service of a variety of high-level goals (e.g., making salad, baking a carrot cake, making stew, making carrot sticks). The knowledge for the high-level task must be provided by the user of the peeler. A breadmaking machine is highly specific in the sense of automating a particular high-level task (i.e., ingredients in; bread out), but includes a variety of subtasks (i.e., mixing, kneading, rising, baking). The machine embodies a great deal of breadmaking knowledge, in fact *substituting* for breadmaking knowledge and skill on

the part of the person using it, but also limiting the variety and quality of bread that can be made. Breadmaking machines are therefore for people who make a lot of fresh bread, but either lack breadmaking skill or, if they have the skill, value convenience over quality.

6. CONCLUSIONS

6.1 Revised Hypothesis

Our findings require extensive revision of our initial four-point hypothesis. In this section, we take each point in turn, discussing how it should be modified in light of our findings.

6.1.1 *Task Specificity.* We began the study thinking that software applications vary along a continuum of task specificity, and we found that it is not that simple. The concept of a *task* as an invariant upon which software designs and behavioral predictions can be based must be replaced with the more complex concept of a *task domain,* within which people's goals and specific tasks vary and, with them, their choice of tools.

Furthermore, we realized that building domain semantics into an application is not simply a matter of degree. Software applications for a given task domain vary in which domain activities they support (not to mention the degree to which their design is based upon task analysis). Software applications also vary in the applicability of their objects and actions across task domains and in whether they provide extension mechanisms, which allow users to add semantics.

We no longer believe that the only way to provide task-specific support is to build extensive domain semantics into a single application. Following the revised hypothesis, we discuss an alternative, perhaps more promising approach.

6.1.2 *Task-Specific Tools Preferred.* We still believe that people would, ideally, prefer software designed specifically to support the task domain they are working in. However, we learned that this is nearly impossible to achieve via single high-semantics applications except in very small, constrained task domains (e.g., household accounting).

We also still believe that the degree of support that software provides for carrying out actions within its users' task domain is more important than whether its user interface is graphical or textual, direct or indirect, menu or command based. Many of today's software applications have user interfaces that merely facilitate manipulation of abstract computation concepts rather than allowing people to work within the task domain.

6.1.3 *Limited Market.* To our claim that more task domain support means a smaller market, we must add the caveat that not all domain knowledge is equal. Building coordination and process semantics into a tool makes it less flexible in the sorts of results it can produce and the sorts of work practices it allows. Thus, software that incorporates that sort of task domain semantics has a more limited market than software that incorpo-

rates other kinds of task domain semantics, e.g., support for specific intermediate work products.

6.1.4 *Factors of Choice.* Our original hypothesis assumed that the primary hindrance to the use of what we were calling task-specific software is acquisition and learning costs. Though it is true that tools people already have and know have a strong advantage, our findings indicate that cost is not the whole story. We found several additional factors that affect what tools will be used to make presentations (see above). To summarize the factors:

— *Speed vs. quality*: How important is it that presentations be fancy versus done quickly? Often, there are two separate production processes: one for most presentations, using tools that optimize the process rather than the product, operated by people possessing less domain expertise, and another for the few fancy or unusual presentations, using tools that optimize the product, operated by task domain experts.
— *Power vs. skill*: How much domain knowledge and skill about slidemaking do prospective users have? Those who have high domain skills apparently prefer tools that stay out of their way and let them exercise their skill. Tools that supply significant domain knowledge are mainly for people who lack either domain skill or the time or desire to exercise their skill (though such tools face the aforementioned cost/benefit hurdle).
— *Interoperability*: Can a tool easily take input from others? Professional slidemakers prefer using collections of tools, each of which provides needed support for certain slidemaking activities. Such people often use dedicated presentation programs as containers and organizers of content produced elsewhere.
— *Support for teamwork*: Does the tool support people working together on a presentation, if that is how presentations are produced at the worksite in question? Most presentation programs assume a single user working alone, but many presentations are created by teams, with different people contributing different parts.
— *Company policy*: What tools does the company prefer, for whatever reason, e.g., history, business relationships, price, familiarity, or standardization?

6.2 Better Task Support through Modularity and Interoperability

How might software developers provide better support for people who are skilled in a task domain, since tools that are intended to be comprehensive within a task domain clearly do not work well for them? Our finding that many professional slidemakers use collections of programs in concert suggests that the most practical approach for software developers may be

to support this way of working intentionally and explicitly. That is, perhaps software developers should provide collections of interoperable tools and services that computer users mix and match as needed to accomplish the goals of the moment, rather than a single tool intended to support a wide range of tasks within a specified domain, end to end.

Modular interoperable tools and services allow not only for task differences, but also for individual user differences, whether in skill level, job type, or personal preference. Computer workers could select and bundle modules according to need and preference into packets that capture regularities in their daily work instead of being faced with trying to use a program having a nonoptimal bundle of services, documented by a fat manual containing mainly irrelevant detail. Local developers [Johnson et al. 1993; Nardi and Miller 1991] might bundle services for individuals or groups or help end-users do that.

The need to support teamwork is a third argument for preferring a set of modular interoperable tools and services over single tools that try to "do everything." With interoperable tools, each contributor to a presentation can use the best—or favorite—tools for his or her part of the presentation; then the work can be put together.

Interoperability is often regarded as the ability to copy data freely between tools, but that is actually only a minimum requirement. A higher form of interoperability is dynamic *sharing* of information between applications and between coworkers. Interoperability of this sort is extremely rare in software for making slide presentations. Proposed open-document protocols from computer and software vendors are intended to allow for this sort of interoperability, but can only do so if application developers avail themselves of the new capabilities.

Even more ambitiously, interoperability can include support for maintaining desired relationships between separately produced parts of a presentation and support for work flow between tools and team members. Our findings suggest, however, that slidemaking tools that were interoperable in this way could succeed with professional slidemakers *only* if the relationships and workflow model were user definable rather than fixed.

The idea of providing collections of interoperable software tools is not new. Nielsen et al. [1986] alluded to it in the conclusions to their study. At that time, the ability of application programs to exchange data with other programs was much lower than it is today, so the most practical way of achieving interoperability then was to put into one large program functionality that otherwise would be in several distinct applications (e.g., word processing, drawing, and spreadsheet). Therefore, Nielsen et al. argued mainly for more multiapplications. However, they quoted one of their informants as saying:

> [Software developers] have a handle on program integration but not task integration. I want task integration. [I] don't care about product integration [Nielsen et al. 1986, p. 167; brackets in original].

Explaining this, Nielsen et al. wrote:

> We believe that what is relevant for professional business users is not necessarily product integration but *task integration*. Task-integration looks at what data and manipulations of that data match the requirements of users' tasks. These task requirements may or may not match the application boundaries of. . .integrated systems [Nielsen et al. 1986, p. 167].

Fischer and Lemke [1988] reached similar conclusions. They suggested that application software would be more usable and useful if it were provided in the form of high-semantics construction kits (e.g., Electronic Arts' Pinball Construction Set™), allowing users to construct applications as needed for specific situations. Even better, according to Fischer and Lemke, is software that provides not only task domain components but also domain-specific guidance in combining them. *Design environments* is their term for such software.

An even earlier, albeit primitive, embodiment of the "interoperable tools" approach is Unix. The primary user interface of the Unix operating system, referred to as the *shell,* is based heavily on a simple form of interoperability: shell users solve information-processing problems by directing streams of data through ad hoc sequences of programs, with each program performing a specific function on the data stream. Unix can be—and has been—criticized soundly for anarchic command naming, cryptic syntax, and general lack of attention to principles of good user interface design [Norman 1981]. Its promoters can also be faulted for attempting to foist it onto task domains and user populations for which it was not designed (e.g., office information processing). However, the fact remains that it has been enormously successful within its original target task domains (software development and technical documentation) and users (software engineers).

What sort of interoperable tools or services might be useful for slidemaking? One example is outlining. Outlining is a service that is useful in many tasks, including slidemaking. Outlining could be selected from a set of services and applied to the task of slidemaking, as appropriate.

6.3 Limitations of Results and Directions for Future Research

Our interest was in exploring the costs and benefits—and computer users' perceptions thereof as manifested in their choice of tools—of task-specific versus task-generic application software. For resource reasons, our study focused on a particular task domain and a limited user population.

Obviously, restricting the study to slidemaking limits the generalizability of our findings. It is perhaps less obvious how our choice of informants limits the generalizability of our results. Though we interviewed business and engineering professionals who create slide presentations infrequently, as well as professional graphic artists whose primary vocation is creating slide presentations, we did *not* interview nongraphic artists who create large numbers of slide presentations (e.g., college professors) or graphic artists who only infrequently create slide presentations. Including such

informants might have helped clarify some of the findings that remain unclear.

Despite the limitations, we feel that our findings do speak to the more general issues raised in our working hypothesis. A focused look at the work of real computer users in a single task domain led us to rethink our key assumptions and to consider an alternative way of supporting end-user application development, i.e., collections of interoperable tools and services.

Of course, the generality of the findings reported here should be validated through comparable studies in other task domains and with other user populations. We believe that the results of the present interview study also suggest the need for in-depth ethnographic research on how people use software in the slidemaking and other task domains. Within the slidemaking task domain, more structured—perhaps experimental—studies would shed additional light on issues such as the tradeoff between task support and learning cost. We hope that this article will stimulate researchers to try to clarify the issue that our findings so successfully muddied, i.e., what it means for application software to provide good versus poor support for its users' tasks. Finally, we hope that this article will stimulate software developers to give more thought to how best to support the work of the people who will use the software.

APPENDIX

A. QUESTIONS ADDRESSED IN INTERVIEWS

1. What is your role in producing presentation slides?
   - 1.1 Do you produce slides yourself or do you supervise others who do it?
     - 1.1.1 What sort of training or experience is required to do the job you do?
     - 1.1.2 [If supervises others] What is the skill level of your employees?
   - 1.2 How much of your total job involves producing presentation slides?
   - 1.3 For whom do you produce these presentation slides?
     - 1.3.1 Who is the customer (i.e., who approves the slides)?
     - 1.3.2 Who is the audience for the presentations?
   - 1.4 What sort of quality level is required for the slides?
     - 1.4.1 How important are elaborate special effects (e.g., animation, dissolve)?
     - 1.4.2 Who decides on appearance and quality, you or the customer?
     - 1.4.3 Are there different kinds of presentations with different quality requirements?
   - 1.5 Do you (your department) follow slide formatting standards?
     - 1.5.1 How do you assure that slides adhere to those standards?
     - 1.5.2 Does your slidemaking software help with standardization of presentations?
2. What software do you use to create presentation slides?
   - 2.1 Who decides what software you use for this?

2.2 Do you use one program or a collection of them?
  2.2.1 [If many] What are the different programs used for?
2.3 Do you use general-purpose drawing software or slidemaking software?
  2.3.1 Why?
2.4 What do you like about each of the programs you use?
2.5 What do you dislike about each one; what would you like to see changed?
  2.5.1 Describe some of the things you do to "work around" limitations of the software.
2.6 How easy is the software for new users to learn?
  2.6.1 How do they learn the software (classes, manuals, using, asking)?
  2.6.2 How did you learn it?
2.7 What other software have you used, tried, or considered for making slides, either here or in previous jobs?
  2.7.1 Why don't you use it now?

3. What is involved in making slides?
3.1 Describe the complete process of producing a presentation, from when you take the assignment to when to deliver it to the customer.
  3.1.1 How much revision is usually required before a presentation is considered done?
3.2 Do you usually create new presentation slides?
  3.2.1 What is hard and what is easy about creating new material—i.e., what goes quickly, and what takes time and work?
3.3 Do you reuse old slides in new presentations?
  3.3.1 What is hard, and what is easy about reusing old material—i.e., what goes quickly, and what takes time and work?
3.4 How do you (your department) organize and keep track of slides and presentations?
  3.4.1 Is each slide a separate file, or are all the slides in a presentation together in one file?
  3.4.2 Do you use directories (folders) and subdirectories (subfolders) to organize your material?
  3.4.3 How do you name your slide (or presentation) files?
  3.4.4 Do you ever fail to find a slide you know you have?
  3.4.5 How does your software hinder you in reusing material?
  3.4.6 How easily can you include a single slide in several different presentations?
3.5 What kinds of revisions are often required in the process of preparing a presentation?
  3.5.1 Which are easy, and which are hard?
  3.5.2 Are the same revisions easy and hard for each of the slidemaking programs you use?
  3.5.3 Some specific cases we'd like to know about:
    3.5.3.1 A slide used in multiple presentations is changed.

3.5.3.2 A company logo or standard border must be added to every slide in a presentation.
3.5.3.3 The order of slides in a presentation must be changed.
3.5.3.4 The round bullets throughout a presentation must be changed to square bullets.
3.5.3.5 The font used throughout a presentation must be changed.
3.5.3.6 Each of the points on a particular slide must be expanded into a separate slide.

B. TABULATED INTERVIEW DATA

Table IV tabulates much of the data gathered in the interviews. It shows the type(s) of software used by *each* informant (or people they supervised) in preparing presentations. It also shows that professional slidemakers tend to distinguish, in the software they use, between preparing "plain" presentations and "fancy" presentations. Finally, Table IV contains an entry for a type of slidemaking professional that our initial hypothesis overlooked: graphics clericals (see row 2).

Abbreviations used in Table IV: presentation = *pres,* drawing = *draw,* painting = *paint,* charting = *chart,* desktop publishing = *pub,* document compiler = *doccomp,* animation = *anim,* spreadsheet = *spread,* database = *dbase,* image processing = *image,* custom = *custm.*

Tabulating the interview data was neither straightforward nor totally objective. If an informant said something like, "I used to use Program X, but I don't anymore," it was not clear whether to include Program X in the software used by the informant. Our tendency was to be inclusive: if an informant used a particular software program for slidemaking until recently, we included it; if he or she abandoned it years before, we excluded it. On the other hand, if an informant said something like, "I've tried Program X, but I don't like it and don't use it," we did not count Program X as being in that informant's tool collection.

It was also not perfectly clear what work processes to tabulate as "working in teams" versus "working alone." In some sense, any situation in which a nongraphic artist has a professional graphic artist produce a presentation involves teamwork: the nonartist sketches or describes the slides; the artist produces a first draft; the nonartist critiques the draft; and the artist revises the presentation until it is satisfactory. In this view, "working alone" includes only cases in which someone creates a presentation for his or her own use. However, we used a more conservative criterion, classifying work processes as "working alone" unless multiple people were working with slidemaking software and sharing datafiles.

If an informant said, "I use Program X, but the people I manage make slides with Program Y," it was not clear whether or how to tabulate Program Y. Our judgment was that software used by slidemaking subordinates should be included in the tables. For this reason, the number of users in some of the tables exceeds the number of informants we interviewed.

Table IV. Presentation-Making Software Used by Informants and Their Subordinates, with Other Pertinent Data

Informant	Graphic Art Skill	Empl. or Freelance	Work Product	Main SW	Aux. SW	Work in Teams?	Who picks SW?
1	pro	empl	fancy pres	pres, anim	unknown	no	self
(1 mngs)	cler(s)	empl	plain pres	pres	none	no	1
2	pro	empl	plain pres fancy pres	draw, pres draw	pub, draw, paint, chart	yes	self
(2 mngs)	pro(s)	empl, free	graphics	none	draw, paint, chart	yes	2
3	pro	free	plain pres fancy pres	pres anim	pub, draw, paint, chart, image	no	client
4	pro	empl	pres	draw	chart, custm	yes	self
(4 mngs)	pro	empl	fancy pres	draw	chart, custm	yes	4
(4 mngs)	pro(s)	empl, free	graphics	none	draw, chart, custm	yes	4
5	pro	empl	plain pres fancy pres	pub, pres, draw anim	draw, paint, image	yes	work group
6	pro	free	plain pres fancy pres	pres anim	paint, image, spread	no	self, client
7	pro	free	pres	draw, pres	pub, chart	no	client, self
8	am	empl	pres	pres	paint, spread	no	self
9	am	empl	plain pres	pub	draw, spread	no	self
10	am	empl	plain pres	doccomp, pub	none	no	self
11	am	empl	plain pres fancy pres	pres pub	none	no yes	self, comp
12	am	empl	plain pres	pub, doccomp	none	no	self, comp
13	pro	free	plain pres fancy pres	pres pub	draw, pub, paint, chart	yes	client, self
14	am	empl	pres	draw	spread, dbase	no	self
15	pro	empl	plain pres fancy pres	pub draw	draw, chart, spread	no	self, comp
16	pro	empl	pres	pub, pres	draw, spread	yes	comp, self

However, for lack of a better rule, we counted references by informants to people they manage as one (1) user regardless of the number of people being referred to.

Note that these tables do not reflect relative frequencies of use of the various types of software. If an informant said that he or she sometimes used a particular program to prepare slide presentations, no matter how infrequently, we included it. Having not asked informants to assign relative usage weights to their slidemaking software tools, we found that we could not objectively extract usage weightings from the interview transcripts. Nonetheless, it was clear to us from the interviews that professional graphic artist informants were much less reliant on dedicated slidemaking programs than our initial hypothesis had predicted them to be. Thus, for example, Table IV does not indicate that, though informant 5

occasionally used a dedicated presentation program, his main tool for presentation making was a desktop publishing program.

C. TRADEMARKS

1-2-3 is a trademark of Lotus Development Corp.
Canvas is a trademark of Deneva, Inc.
Charisma is a trademark of MicroGrafx.
CricketGraph is a trademark of Computer Associates.
Corell Draw is a trademark of Corell, Inc.
DeltaGraph is a trademark of DeltaPoint, Inc.
Excel, PowerPoint, and Word are trademarks of Microsoft Corp.
Frame Maker is a trademark of Frame Technology Corp.
Harvard Graphics is a trademark of Software Publishers Corp.
Illustrator and Photoshop are trademarks of Adobe, Inc.
MacPaint, MacDraw, and MacDraw Pro are trademarks of Claris, Inc.
Macromind Director is a trademark of MacroMedia, Inc.
Persuasion and PageMaker are trademarks of Aldus Corp.
Pinball Construction Set is a trademark of Electronic Arts.
Unix is a registered trademark in the United States and other countries, extensively licensed through X/Open Company, Ltd.
Ventura Publisher is a trademark of Xerox Corp.
Word Perfect is a trademark of Word Perfect, Inc.

ACKNOWLEDGMENTS

The research described herein was conducted while the authors were employed at Hewlett-Packard Laboratories, Palo Alto, California. We thank Michelle Gantt, the anthropology student intern and graphic artist who served as the interviewer for the study and—by questioning our initial hypothesis about who uses task-specific versus generic software—as the primary catalyst for our conducting it. We thank Jim Miller and Craig Zarmer, our colleagues on the ACE project, for their ideas and for comments on earlier drafts of this article. We also thank Jonathan Grudin and several anonymous reviewers for comments and suggestions that helped us improve this article. Last but not least, we thank our informants for taking time out of their busy schedules to educate us about their work practices.

REFERENCES

CASNER, S. 1991. A task-analytic approach to the automated design of graphic presentations. *ACM Trans. Graphics, 10,* 111–151.

FISCHER, G. AND LEMKE, A. C. 1988. Construction kits and design environments: Steps toward human problem-domain communication. *Hum. Comput. Interaction 3,* 3, 179–222.

FISCHER, G., MCCALL, R., AND MORCH, A. 1989. Design environments for constructive and argumentative design. In *Proceedings of the ACM Conference on Computer-Human Interaction (CHI'89).* ACM, New York, 269–275.

FOX, R. 1995. NewsTrack. *Commun. ACM 38,* 5 (May), 9.

GOULD, J., BOIES, S., AND LEWIS, C. 1991. Making usable, useful, productivity-enhancing computer applications. *Commun. ACM 34,* 1 (Jan.), 75–85.

HUTCHINS, E., HOLLAN, J., AND NORMAN, D. 1986. Direct Manipulation Interfaces. In *User Centered System Design*, D. Norman and S. Draper, Eds. Erlbaum Associates, Hillsdale, N.J.

JOHNSON, J. A., NARDI, B. A., ZARMER, C. L., AND MILLER, J. R. 1993. ACE: Building interactive graphical applications. *Commun. ACM* 36, 4 (Apr.), 40–55.

LEWIS, C. AND OLSON, G. 1987. Can principles of cognition lower the barriers to programming? In *Empirical Studies of Programmers: Second Workshop*. Ablex, Norwood, N.J., 248–263.

MYERS, B. A. 1989. User interface tools: Introduction and survey. *IEEE Softw. 6*, 1.

NARDI, B. A. 1993. *A Small Matter of Programming: Perspectives on End User Computing*. MIT Press, Cambridge, Mass.

NARDI, B. A., Ed. 1996. *Context and Consciousness: Activity Theory and Human-Computer Interaction*. MIT Press, Cambridge, Mass.

NARDI, B. A. AND MILLER, J. R. 1990. The spreadsheet interface: A basis for end-user programming. In *Proceedings of Interact'90*, D. Diaper, D. Gilmore, G. Cockton, and B. Shackel, Eds. Elsevier Science, New York.

NARDI, B. A. AND MILLER, J. R. 1991. Twinkling lights and nested loops: Distributed problem-solving and spreadsheet development. *Int. J. Man-Machine Stud. 34*, 161–184.

NARDI, B. A. AND ZARMER, C. L. 1993. Beyond models and metaphors: Visual formalisms in user interface design. *J. Visual Lang. Comput. 4*, 5–33.

NIELSEN, J., MACK, R. L., BERGENDORFF, K. H., AND GRISCHKOWSKY, N. L. 1986. Integrated software usage in the professional work environment: Evidence from questionnaires and interviews. In *Proceedings of the ACM Conference on Computer-Human Interaction (CHI'86)*. ACM, New York, 162–167.

NORMAN, D. 1981. The trouble with Unix. *Datamation 27*, 12 (Nov.), 139–150.

OLSEN, D., MCNEILL, T., AND MITCHELL, D. 1992. Workspaces: An architecture for editing collections of objects. In *Proceedings of the ACM Conference on Computer-Human Interaction (CHI'92)*. ACM, New York, 267–272.

PRIETO-DIAZ, R. 1990. Domain analysis: An introduction. *Softw. Eng. Notes 15*, 2 (Apr.), 47–54.

SPA. 1995. Home users: Fourth annual survey. Software Publishers Association, Mar. 14. *Reported by Internet newswire.*

VLISSIDES, J. AND LINTON, M. 1990. Unidraw: A framework for building domain-specific graphical editors. *ACM Trans. Inf. Syst. 8*, 3 (July), 237–268.

ZARMER, C. L. AND JOHNSON, J. A. 1990. User-interface tools: Past, present, and future trends. Hewlett-Packard Laboratories Tech. Rep. HPL-90-20, Hewlett-Packard, Palo Alto, Calif.

ZARMER, C. L., NARDI, B. A., JOHNSON, J. A., AND MILLER, J. R. 1992. ACE: Zen and the art of application building. In *Proceedings of the 25th Hawaii International Conference on System Sciences (HICSS-25)* (Koloa, Hawaii, Jan. 7–10). Vol. 2. ACM, New York, 687–698.

Received April 1994; revised June 1995