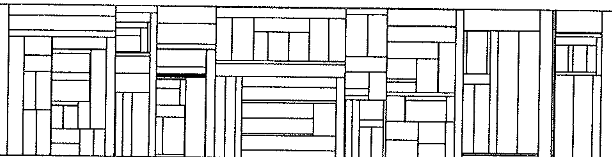
ISSN 0105-8517

Systematic Sources of Sub-Optimal Interface Design in Large Product Development Organizations

Jonathan Grudin

DAIMI PB – 321
July 1990

COMPUTER SCIENCE DEPARTMENT
AARHUS UNIVERSITY
Ny Munkegade, Building 540
DK-8000 Aarhus C, Denmark



SYSTEMATIC SOURCES OF SUB-OPTIMAL INTERFACE DESIGN IN LARGE PRODUCT DEVELOPMENT ORGANIZATIONS

Jonathan Grudin

Department of Computer Science, Aarhus University

INTRODUCTION
- Product Development Organizations
- Development Projects
- Interface Development and User Involvement

PART 1. OBSTACLES TO USER INVOLVEMENT
- Challenges in Motivating Developers
- Challenges in Identifying Appropriate Users
- Challenges in Obtaining Access to Users
- Challenges in Motivating Potential Users
- Challenges in Benefiting from User Contact
- Challenges in Obtaining Feedback from Existing Users
- The Difficulty of Identifying the Design Team
- Software Development Processes Developed for Other Purposes
- The Routinization of Development
- *Positive* Conditions for User Involvement in Product Development
- Overcoming the Obstacles

PART 2. WITHOUT KNOWLEDGE OF USERS: DESIGN BY DEFAULT
- Software Development Goals
- Cognitive Processes and Individual Goals
  - Individual Goals Arising from Professional Responsibilities
    - The Goal of Understanding the Software Architecture
    - The Goal of Design Simplicity
    - The Goal of Design Consistency
    - The Goal of Anticipating Low-Frequency Events
    - The Goal of Thoroughness
  - Individual Goals Arising from Personal and Career Issues
    - The Goal of Retaining Responsibilities
    - The Goal of Extending Skill Repertoire
    - The Goal of Personal Expression
- Social Processes and Group or Team Goals
  - The Goal of Communication
  - The Goal of Coordination
  - The Goal of Compensation
  - The Goal of Cooperation
- Organizational Processes and Corporate Goals
  - The Goal of Division of Labor
  - The Goal of Efficient Decision-Making
  - The Goal of Organizational Survival

CONCLUSION: MULTIPLE CONSTRAINT SATISFACTION
- Redefining the User Population
- Strengthening the Use of Mediators
- Organizational Change
- Technology Support

INTRODUCTION

The need for the developers of interactive systems to understand the eventual users and their work is well known. The difficulty of acquiring that knowledge indirectly has led to an emphasis on direct user contact during the development process. Although good human-computer interfaces can be found, the difficulty of developing them is substantial. This paper explores the underlying problems in one systems development context: large product development organizations. Most of these companies were formed before the human-computer interface attained its present prominence. As a result, in establishing their organizational structures and development processes, little or no consideration was given to the particular needs of interface development.

The first part of the paper outlines common difficulties in achieving and benefiting from user involvement in development. The second part describes organizational goals that can conflict with good interface design. These goals often seem unrelated to interface considerations, but in the absence of knowledge of users and their work, they may be an unrecognized, subtle source of bad design decisions. To overcome such organizational constraints and forces may ultimately require organizational change; those now working within such organizations must be aware of the problems and seek constructive paths around them.

The paper draws on the growing literature in the field of human-computer interaction, much of which originates in product development companies. It also draws on surveys and interviews of over 200 interface designers in several product development companies (Grudin and Poltrock, 1989; Poltrock, 1989a), my experiences in product development, and thousands of conversations with fellow developers over the years. Of course, organizations vary considerably. Reliable, industry-wide data are difficult to find. The obstacles described here are encountered, but not universally. The hope is that the forewarned reader will be better able to anticipate, recognize, and respond to these and similar challenges when they appear.

Product Development Organizations

Our focus is on large companies that develop and market interactive software systems and applications: systems with a human-computer dialogue or interface. Projects resulting from specific contracts, and internal development groups within large organizations, are not addressed. Contract and internal development, which do not result in externally marketed products, have different advantages and disadvantages in developing interactive systems (Grudin, 1990d). Of course, a company may straddle categories: a product development company may bid on government contracts, an internal development group may decide to market a system built initially for internal use, and so forth. In addition, *small* product development companies may not experience the problems described here, while companies of moderate size may experience some and not others.

Although product development accounts for only a fraction of interactive systems development, it is an influential fraction. Large product development companies are visibly concerned with usability and "look and feel." They have hired and trained many user interface specialists, recruiting heavily from research universities. These specialists dominate the conferences and journals in the field of human-computer interaction, especially in the United States.

These companies matured in the 1960s and 1970s; making their money selling or leasing hardware. Software functionality was secondary and the human-computer interface received little attention. Since then, software has come to rival hardware in importance -- many of the successful new product development companies of the 1980's primarily sold software. The focus was on functionality and price, not on the interface -- until the success of the Macintosh in the late 1980s. Now, the interface is increasingly important.

Attitudes may be changing, but the current business operations and development procedures of most large product development companies were formed when hardware and software functionality were the only considerations. It is therefore not surprising that existing organizational structures and processes do not facilitate interface development. In fact, they often systematically *obstruct* the design and development of good interfaces.

Development Projects

The process of defining a product can be separated into two parts: the events before the project is launched and those occurring during development. An ongoing development project can generally provide a time line showing a start date, a projected completion date, and a few milestones along the way. Reality is usually somewhat less precise than that. One event flows into another. Rather than a decision being made at one point in time, those involved may gradually realize that a decision had emerged sometime earlier. Nevertheless, teams are formed, assignments announced, budgets allocated. Some projects use a number based on the month and day of initiation as their first working name.

Using this demarcation point, the product definition, consisting of the high-level functionality to be developed or implemented, precedes the project start date. While implementing that functionality, the team designs and develops the necessary low-level functions and other aspects of the human-computer dialogue or interface. It is notoriously difficult to draw a line between software functionality and its "user interface," and potential users have a substantial interest in *both*. However, we can distinguish between the high-level description of the product that precedes project initiation and the low-level aspects of the design that are worked out later. This paper will generally apply the expressions "human-computer interface" or "interface" to the latter. For example, the interface to a word processor includes not only the "look and feel" but also whether a one-step "move" or a two-step "cut" and "paste" is provided. In addition, the "interface" is defined broadly to include documentation, training, hot-line support, or other elements that directly affect the users' experiences.

Different groups are involved in these two phases. A management group, with representation from development, works out product definition. Companies vary in the degree to which they are driven by their engineering or marketing divisions, but marketing often has a particularly strong influence at this stage. Once the project is assembled, this group may recede from view, monitoring progress through documentation and management reviews.

Potential product users are sometimes involved in each phase. During product definition, market research and focus groups may provide contact. Typically, these will seek out *customers*, rather than users --

information system specialists from large companies whose job is to represent user requirements (as well as management requirements and other factors influencing purchase decisions). During development, experimental studies of design alternatives and prototype testing may involve users.

An interesting question is whether greater user participation in product definition would be useful. What form of user involvement might be optimal at that stage? Better to sample a wide range of potential users or to work closely with a small number? Open-ended needs-finding -- involving users without any preconceptions -- is rare, though not unheard of. It might lead to good product ideas, but other forces are at work defining products. Most companies operate within a restricted product range to begin with. They generally have existing products that are in clear need of improvement. Indirect channels through sales, marketing, and consultants are a source of product ideas. Other ideas are formulated by responding to competition and monitoring technological innovation. A dearth of product ideas is not usually a problem.

A second question is whether user participation might continue across both the product definition and product development phases. Given the way many product development companies operate, the answer is no -- there is remarkably little carryover of the *company's* personnel from one phase to the next. This may be based on the contract model of system development -- the belief that a written requirements specification is enough to communicate a product idea. It may also lead to some of the faults of contract development: a product being delivered by developers that is not quite what those defining it had in mind.

While there may be room for experimentation, product development does generally begin with a product idea defined primarily in terms of high level functionality (although as the interface gains prominence, attempts to define it will move forward in time, creating new challenges). This paper concentrates on the period after the baton is passed to the development team; in particular, to the process of defining and developing the human-computer interface.

Interface Development and User Involvement

Product developers can acquire an understanding of computer users without making direct contact. Many mediators exist to facilitate the flow

of information between computer users and developers. These include marketing and sales organizations, consultants, information systems specialists, users groups, standards organizations, trade magazines, and journals. Of course, the calls for direct user involvement question the effectiveness of these intermediaries -- and the demands on them are escalating.

As computer users become less technical and more diverse, developers have more need to obtain information about them and their work environments. Also, as competition increases in some markets, increased knowledge about users is needed to fine-tune products. Finally, emerging applications that support groups ("groupware") will require *more* information about users and their environments than did single-user products. Most groupware must support a wider range of users and a greater percentage of users in a given setting. The limitations of any one person's intuitions for group behavior also require that more information be gathered during design and development. Information about work environments is also needed to support product adoption, where potentially deadly problems must be headed off, problems unfamiliar to developers of single-user applications (Grudin, 1990c). In short, groupware brings product developers into much the same situation that internal developers have been in from the start: obligated to examine individual and role differences. All of these support the view that much more information about users must reach developers. Mediators may be unable to meet the challenge of providing this information, necessitating direct contact between developers and potential users.

The most direct and extensive form of user engagement is participatory or collaborative design, enlisting potential users as full members of the design team (e.g. Bjerknes, Ehn and Kyng, 1987). A limited form of this is obtained by hiring people from user organizations to work in development groups as "domain experts." More common but also more circumscribed user involvement is provided by limited-duration studies of existing or potential users, or individuals presumed to be much like them.

In the context of product development, Gould and Lewis (1983) made an early, forceful argument for participatory design, eschewing reliance on mediators and more limited empirical approaches. They wrote "We recommend that <u>typical users</u> (e.g., bank tellers) be used, as opposed to a 'group of expert' supervisors, industrial engineers, programmers. We

recommend that these potential users become part of the design team from the very outset when their perspectives can have the most influence, rather than using them post hoc to 'review,' 'sign off on,' 'agree' to the design before it is coded." Their message has been repeated in several prominent papers since then (e.g., Gould and Lewis, 1985; Gould, Boies, Levy, Richards and Schoonard, 1987; Gould, 1988) and is widely cited, yet it is rarely adopted. Gould and Lewis (1985) allude to unexplored "obstacles and traditions" that stand in the way. This paper explores those obstacles and traditions.

PART 1. OBSTACLES TO USER INVOLVEMENT

The inherent nature of product development, in which a broad market is sought and the actual users are not known until after development is completed, presents challenges to involving users effectively. Additional obstacles to bringing developers into contact with existing or potential users can be traced to the division of labor within the organization (see Figure 1).[1] Existing assignments of responsibility may serve useful purposes, but they separate software developers from the world outside. With non-interactive systems this was not a major problem, but it is a problem now and is likely to get worse as user requirements and expectations increase. Contact with customers and users is the province of groups or divisions outside of development: sales, marketing, training, field support, and perhaps upper management. The people assigned these tasks are not primarily concerned with the interface, their relevant knowledge is not systematically organized, and they are often located far from the developers. They have a limited sense of what information would be useful or to whom to forward it. After discussing these structural impediments, we will examine difficulties that can be traced to standard software development procedures and techniques.

---

[1] Many small and large variants of this organizational structure are found. To take one example, functions such as Quality Control and Performance Analysis may be handled centrally.

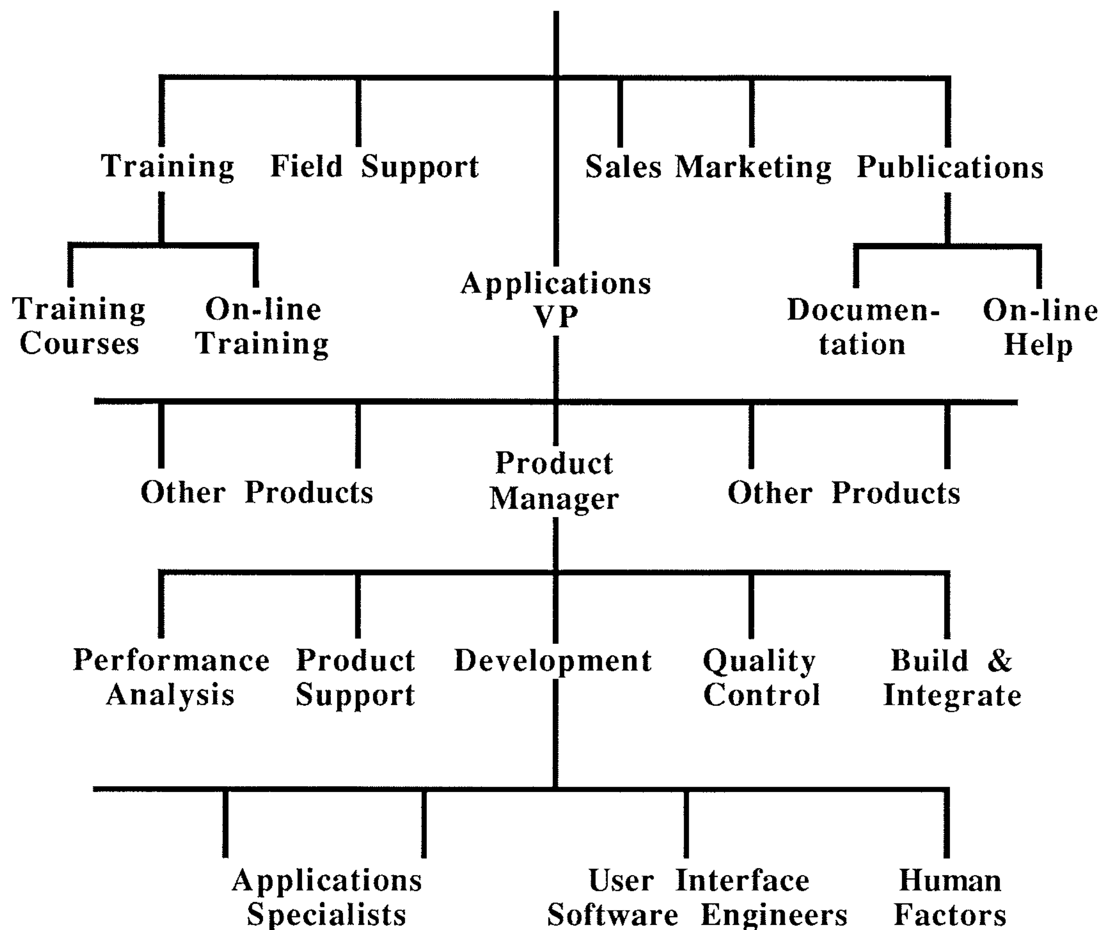


Figure 1. Organization chart showing separation of user-related functions.
Adapted from Poltrock (1989c).

Challenges in Motivating Developers

Success may require most or all members of the development team to commit to user involvement. One person can work with users and try to introduce the results into the process, but iterative development requires a broader commitment, prototyping and testing may require software support, the results have to be valued, and so forth. Management must be willing to invest the resources, and the help of others may be needed to smooth contacts with users.

Although most developers would agree to user involvement in principle, it requires a greater commitment to make it work. Engineers and other team members may not follow through for several reasons. They may lack empathy or sympathy for inexperienced or non-technical computer

users. When developers and users meet, they may find that different values, work styles, even languages get in the way of communicating. Developers tend to be young, rationalistic and idealistic, products of relatively homogeneous academic environments. They often have little experience or understanding of the very different work situations and attitudes of many system users. The best of intentions can succumb to these factors, especially in the face of the slowness and imprecision that often accompanies user involvement.

Challenges in Identifying Appropriate Users

Developers may have a market in mind, but the actual users of a product are not known until the product is bought. The fate of many products, both negative and positive, are reminders of the inherent uncertainty in product development. The IBM PC is an example of a product with wider than expected appeal, while we can be confident that the designers of countless failed products anticipated users who never materialized.

Further obstacles to identifying potential users stem from the nature of developing products intended to appeal to a broad range of people. The effort is focused on casting as wide a net as possible; reversing gears to try to identify specific or characteristic users is difficult. Choosing one may seem to eliminate other possibilities. The seriousness of the problem of defining characteristic users can be seen by considering the experience of Scandinavian researchers in a different, *more favorable* development context. Based on the participatory design approach described above, these projects began with relatively constrained user populations, within one industry or even one organization. Even so, selecting "representative" users could be a major challenge (e.g., Ehn, 1988, pp. 327-358). Such problems are greater for developers of generic products.

Obstacles also arise from the division of labor. User interface specialists rarely have "the big picture." They may work with a development team assigned to a single application or even to part of an application. Not even the project manager has a perspective encompassing the application mix that customers are expected to use, the practices and preferences of the installed customer base, and strategic information about the intended market for a product. This broad perspective may be found in Marketing or Sales divisions, which are often geographically and organizationally distant from the development groups. The projected market -- the

identity of the future users -- may be closely guarded by upper management due to its competitive importance.

In large companies, marketing and sales representatives become species of "users" of products emerging from development. They also consider themselves to be internal advocates for the customers. Since the customers are often information specialists or managers, rather than "end-users," the chain of intermediaries lengthens. Low levels of contact and mutual respect between marketing and development (e.g. Kapor, 1987; Grudin and Poltrock, 1989; Poltrock, 1989a) can further reduce the value of this very indirect link between developers and users.

Another complication in identifying appropriate users is that a system is often modified substantially after the development company ships it but before the users see it. This is done by software groups within customer organizations and by "value-added resellers" who tailor products for specific markets, for example. These developers are in a real sense "users" of the product -- perhaps among the most important potential users. It may be more appropriate for *them* to involve the actual "end-users". In any case, the initial development team must discover which aspects of their design are likely to be "passed through" to users. Third-party intermediaries represent an opportunity, but their role also complicates the selection of "representative end-users."

Challenges in Obtaining Access to Users

Once candidates have been identified, the next challenge is to make contact with them. Obstacles may arise within either the users' organization or the development organization.

Contacts with customers are often with managers or information system specialists, rather than with the computer users themselves. Getting past them may not be easy: their job is precisely to represent the users. In addition, the employers of prospective users may see little benefit in freeing them to work with an outside development group.

Within the product development company, *protecting (or isolating) developers from customers is traditionally a high priority*. The company cannot afford to let well-intentioned developers spend all of their time customizing products for individual users -- priority is given to developing generic improvements to benefit scores or hundreds of users. Savvy customers are well aware of the value of having the phone number

of a genial developer. And barriers erected to keep users from contacting developers also prevent developers from easily connecting with users: the relationships and channels are not there.

The development company's sales representatives may be reluctant to let developers meet with customers. A developer, coming from a different culture, might offend or alarm the customer, or create dissatisfaction with currently available products by describing developments in progress. Similarly, Marketing may consider itself to be the proper conduit into the development organization for information about customer needs and may fear the results of random contacts between developers and users. In one company, developers, including Human Factors Engineers, were prevented from attending the annual Users' Group meeting. Marketing viewed it as a show staged strictly for the customers.

Sometimes the potential user is *within* the development company. Convenient, but a dangerous special case to rely on. The company is not in business to build products for itself, and user environments are growing less likely to resemble development environments.

Challenges in Motivating Potential Users

In an in-house development project, the product developers share the same management (at some level) with the potential users. This is not true in the case of product developers and external users, which may make it more difficult for the users to obtain time away from their jobs. In addition, the potential users may be less motivated, knowing that they may not end up actual users of the final product. The problems of sustaining user involvement have been recognized as substantial even in internal development projects. Of course, for contacts of limited duration, many computer users and their employers are pleased to be consulted.

Potential users may be less motivated if they do not see how the planned product will benefit themselves if they *do* get it. This is particularly a problem for large systems and for groupware applications, most of which require widespread use but selectively benefit managers (Grudin, 1988; 1990a). The situation is even worse if the potential users feel that their jobs might be threatened by a product leading to increased efficiency.

Challenges in Benefiting from User Contact

Given an uncertain identity of future users and a wide range of conceivable candidates, assessing one's experiences with a small number of possible users can be difficult. The Scylla of over-generalizing from a limited number of contacts is accompanied by the Charybdis of bogging down when users disagree. Finally, if user involvement succeeds in producing design recommendations, the work has just begun. Design ideas, whatever their source, must be steered through a software development process that is typically fraught with obstacles to interface optimization. User involvement may increase the odds of successfully navigating this course, but the journey is rarely easy, for reasons described below.

Challenges in Obtaining Feedback from Existing Users

Feedback from users may be collected informally or through bug reports and design change requests. The latter generally focus on the basics and on what is of primary importance in the marketplace -- e.g., hardware reliability and high-level software functionality -- and not on interface features. Even the little information that *is* collected rarely gets back to developers. Field service or software support groups shield developers from external contacts by maintaining products and working with customers on specific problems. The original product developers move on to new releases or product replacements, are reassigned to altogether different projects, or leave the company for greener pastures.

The extent of feedback may vary with the pattern of marketing and product use. A company such as Apple, with a heavy proportion of "discretionary" purchases initiated by actual users rather than by management or information systems specialists, *benefits* from having a particularly vocal user population. In general, though, the lack of user feedback may be the greatest hindrance to good product interface design and among the least recognized defects of standard software development processes. System developers cannot spend *all* of their time fielding requests from customers, but their overall lack of experience with feedback is an obstacle both to improving specific products and to building an awareness of the potential value of user participation in design. Developers rarely become aware of the users' pain.

This point deserves emphasis. Engineers are engaged in a continuous process of compromise, trading off among desirable alternatives. Interface improvements will be given more weight if engineers are aware of the far-reaching, lasting consequences of accepting an inferior design. Consider some typical tradeoffs: "This implementation will save 10K bytes but be a little less modular." "This design will run a little faster but take a month longer to complete." "This chip provides two more slots but adds $500 to the sales price." Each requires a decision. Once the decision is made, the price in development time, memory size, or chip expense is paid and the matter is left behind. In this environment, the interface is just one consideration among many. "This interface would be nicer, but take two months longer to design." Without feedback from users, once this decision is made, it too can be forgotten. The decision may adversely affect thousands of users daily for the life of the product, but the developer remains unaware of this. The interface *is* special, but developers do not recognize that -- once it is built and shipped, they are on to the next job, and other people (including users) must do the sweeping up.

The Difficulty of Identifying the Design Team

User involvement would be easier if one group had responsibility for all aspects of usability, as recommended by Gould (1988). But the "user interface," broadly defined, is not often the province of one recognizable team in a large product development company. The hardware is designed by one group, the software by another, the documentation by a third, and the training by a fourth. Representatives from other groups may have periodic involvement -- reviewing design specifications, for example. A product manager with little direct authority over developers may coordinate scheduling. Individuals from several different marketing groups, such as competitive or strategic analysis[2] and international marketing, may contribute. Members of support groups such as human factors or performance analysis may participate, although not necessarily throughout the project. Several levels of management may monitor the process and comment at different stages. In concert, these people contribute to defining a computer user's experience with the system or

[2] Competitive analysis may seem to be a logical ally of a development organization. However, in practice their concern may be the effective marketing of existing products against competition, rather than the planning of future products.

application, yet communication among them may be surprisingly sparse. With whom are users to participate? In addition, turnover in project personnel is common, a further obstacle to sustained user involvement.

Matrix management is one approach to overcoming organizational separation. Representatives of a subset of these groups are given temporary assignment to a project. However, due to the perception or the reality that the contribution of "support roles" is limited to certain phases of a project, such assignments are often of limited duration or effectiveness. (An example of a matrix management effort succumbing to these forces is described in Grønbæk, Grudin, Bødker and Bannon, 1990.)

The general neglect of on-line help illustrates how divided responsibility can affect interface design. A good help system might save the company a substantial amount of money in customer "hand-holding," service calls, printed documentation, and so forth. The savings would be in the budget of, say, the Customer Service Department. But the effort and expense would have to come from Development, who may get more credit for devoting their resources to new functionality instead. Missing is the "affirmative action" needed to promote on-line help in the face of possible lack of developer empathy with less experienced users reenforced by the lack of mutual contact. Thus, help systems often end up with a low development priority.

Software Development Processes Developed for Other Purposes

This section turns from the structural aspects of product development organizations to consider the influences on interface development of some widely used software development processes and methods. These were developed when interactive systems were rare. Many were developed in the context of contract or internal development projects, not product development. As a result, they may obstruct rather than facilitate the development of usable interactive systems.

To begin with, computing resources were too expensive to devote much to the interface. Nor was the demand great: most computer users were engineers who understood the system or were willing to learn it. The computer use environment was similar to the computer development environment. Thus, the interface that emerged during development and debugging was often adequate or even *appropriate* for use -- when the

interface reflects the underlying architecture the engineer need learn only one model.

One legacy of this era is the persistence of the belief that the interface can be ignored or tidied up at the end of development. Late involvement in the software development process is a common complaint of members of support groups such as human factors and technical writing (Grudin and Poltrock, 1989). They are the project members most likely to advocate user involvement. If management is unaware of the need for *their* early and continual involvement, how much support will their calls for early and continual *user* involvement receive?

Over time, the original "engineering interfaces" are being replaced by interfaces developed for increasingly diverse user populations, following a pattern seen in other maturing technologies (Gentner and Grudin, 1990). Software development methods did not anticipate this change. New approaches to development are emerging (e.g., Boehm, 1988; Perlman, 1989), but have yet to be proven or widely adopted. One source of inertia is that insufficient information about user environments reaches developers: the degree to which development and use environments have diverged is not appreciated.

"Waterfall" models of software development arose in the context of large government projects. By their nature, competitively bid contracts emphasize written specifications: contact with the eventual users may be forbidden or discouraged following the initial requirements definition, occurring prior to the selection of the developers. Separate contracts may be let for system design, development, and maintenance. Approaches to development emerging from this tradition included structured analysis, where the task "establish man-machine interface" is relegated to one sub-phase of system development (De Marco, 1978), and Jackson System Development, which "excludes the whole area of human engineering in such matters as dialog design... it excludes procedures for system acceptance, installation, and cutover," (Jackson, 1983). Because such methods do not specify user involvement in design, project plans do not anticipate it. Development organizations are not structured to facilitate it and often work against it. This is obviously not ideal for interactive systems development, where early and continual user involvement has been an early and continually recommended principle for developing usable systems.

A "Catch 22" is that even *late* user involvement is blocked. Once the underlying software code is frozen, a fully functioning system is available for user testing -- but at that very moment, documentation moves into the critical path toward product release. Since it is the software interface that is being documented, the interface is also frozen -- before a user can try it out!

Prototyping and iterative design are recommended by every developer cited here as proposing innovation in technique or methodology -- Bjerknes et al., Boehm, Ehn, Gould and Lewis, Perlman, as do many unnamed. These go hand in hand -- there is little point to prototyping if the design cannot be changed. Unfortunately, the high visibility of the interface works against iterative design in three ways: i) the interface is grouped with aspects of the product that must be "signed off" on early in development; ii) support groups, such as those producing documentation, training, and marketing, are strongly tied to the software interface and affected by changes; c) iteration or change in the interface is noticed by everyone, which can create uneasiness, especially in an environment with a history of stressing early design.

The emphasis on careful early design makes sense for non-interactive software, with its relatively predictable development course. It works less well for the interface, where design uncertainty is inevitable -- the motivation for prototyping and iterative design in the first place. As the interface grows in importance, the desire to see it alongside the proposed functionality in the preliminary design will only grow. And once management has "signed off" on a design, changes require approval. Poltrock (1989b) observed the unique problems that high visibility and dependencies create for the interface development process. One developer summed it up: "I think one of the biggest problems with user interface design is that if you do start iterating, it's obvious to people that you're iterating. Then people say, 'How is this going to end up.' They start to get worried as to whether you're actually going to deliver anything, and they get worried about the amount of work it's creating for them. And people like (those doing) documentation are screwed up by iterations. They can't write the books. Whereas software, you can iterate like mad underneath, and nobody will know the difference."

Interface development is distinct from other software development. Gould and Lewis (1985) summarized it this way: "'Getting it right the

first time' plays a very different role in software design which does not involve user interfaces than it does in user interface design. This may explain, in part, the reluctance of designers to relinquish it as a fundamental aim. In the design of a compiler module, for example, the exact behavior of the code is or should be open to rational analysis... Good design in this context is highly analytic, and emphasizes careful planning. Designers know this. Adding a human interface to the system disrupts this picture fundamentally. A coprocessor of largely unpredictable behavior (i.e., a human user) has been added, and the system's algorithms have to mesh with it. There is no data sheet on this coprocessor, so one is forced to abandon the idea that one can design one's algorithms from first principles. An empirical approach is essential."

Solutions to these problems can be found -- *will* be found -- but the problems are new and adopting the solutions will require changing the way we work. Unfortunately, an innovative process proposal is unlikely to leave management as comfortable as a detailed product design specification.

The Routinization of Development

As competition and the pace of change increase, product development companies are pressured to turn out enhancements and new products in a timely, predictable fashion. Consider this analysis: "Ashton-Tate's decline began with what is becoming a well-worn story in the industry: failure to upgrade a market-leading product. Dbase III Plus went for almost three years before being upgraded, while competitor's products were upgraded as often as twice in that time," (Mace, 1990). A similar pattern of predictable new releases is found in other maturing markets, from automobiles to stereo systems. The result is pressure for a predictable and controllable software development process: for routinization of development. Parker (1990) describes a perceived solution to the problem described in the previous quotation: "Lyons (an Ashton-Tate executive) responds that he can keep customers by providing predictable if not always exciting upgrades. 'Customers don't want to be embarrassed; they want their investment to be protected. If you are coming out with regular releases, even if they skip a release because a particular feature is missing, they will stay (with the product) because the cost of change is large.'"

This perceived need for controlled development creates difficulties for design elements or approaches that have uncertain duration or outcome. Interface design in general has a relatively high level of uncertainty, and user involvement can increase development time and introduce the possibility of changing its direction. This is the intent, of course -- to produce a better design -- but it nevertheless works against these powerful pressures for predictability.[3]

Positive Conditions for User Involvement in Large Product Development Companies

To end on a note of optimism, consider that these companies also provide some *support* for involving users in interface design, primarily through putting the interface itself in the spotlight.

A better interface is one way to distinguish a product and to increase its acceptance in a competitive marketplace. Applications are reaching out to "discretionary" users, people who have the choice of whether or not to use a computer, and the greater availability of alternatives further increases buyer discretion. Computer users are likely to consider usability in exercising discretion.

Large product development organizations often have considerable resources to devote to usability -- development costs are highly amortized. As noted above, these companies are major employers of human factors engineers and interface specialists.

There is a positive side to the relatively frequent upgrades and product replacements: developers can break out of "single-cycle" development. Evaluation of existing product use can feed into the design of later versions and good ideas arriving too late for use on a specific development project can be retained for later use, (Grudin, Ehrlich, and Shriner, 1987).

Product development efforts may have a large supply of potential users, and the fate of a product doesn't depend so heavily on the situational factors that operate in any one given site.

---

[3] Friedman (1989) discounts claims of a trend toward routinization and deskilling of programming. Perhaps he is right, but his focus is on internal software development centers in large corporations, not on software product development. The competitive and marketing pressures that might lead to efforts to increase control of development are less evident in the environments he studies.

Finally, while inertia may develop, software product development companies were founded on change and recognize at a deep level that they must change to survive, leading to some openness to experimentation.

Overcoming the Obstacles

To one working within a large product development organization, the obstacles sometimes seem insurmountable. But as just noted, the company has a powerful incentive to improve product interfaces. Ease of learning and use becomes a more important marketing edge as software products mature. Adding a new bell or whistle may not help much if the already available functionality is underutilized. In addition, declining hardware and software costs permit more resources to be directed to the interface. These forces have already pushed large product development companies into the forefront of human factors research and development in the United States. In the long term, organizational structures and development processes may evolve, institutionalizing solutions to the problems described here. The forces in development companies that work systematically *against* user involvement stand in the way of product optimization and success.

The directions that these companies will take are not obvious. As the focus of development shifts from generic products to systems and applications that meet the needs of different specific markets, companies may have to choose between working closely with independent developers, working with value-added resellers who in turn work with "end-users," or working with the diverse computer users themselves. Each alternative will benefit from "user involvement" -- but the identity of the "users" will vary.

In the meantime, where can developers look for approaches to overcoming these obstacles? Persistence should not be underestimated. There are examples of successful case studies and general approaches, such as the development of the Olympic Message System at IBM (Gould et al., 1987) and the contextual interview approach at Digital (Whiteside, Bennett, and Holtzblatt, 1988).

Another promising source of new techniques is the experience derived from other development contexts, notably European projects based more on internal or in-house development. In part because some of the obstacles described in this paper are encountered in muted form or not at

all in such contexts, user involvement is more often achieved, resulting in elaboration of issues and techniques. Several of these projects and approaches are described in the Proceedings of CSCW'88; see also Bjerknes et al. (1987). Of course, transferring what has been learned from internal development to product development will not always be easy or even possible.

The UTOPIA project (described in Ehn, 1988) explicitly applied some of these approaches to a product development effort. In this project, a small set of potential users was heavily involved with the developers, while techniques including a newsletter were used to involve a much broader selection of potential system users on a more limited basis during design and development.

Experiments with prototype testing and iterative development are increasing our understanding of when and how they are most effectively used. Boehm's (1988) spiral model of development builds these techniques into a disciplined software engineering methodology. He is one of several writers encouraging an explicit change of focus in development from the current "product focus" to a focus on process *per se* in development. Grønbæk et al. (1990) describe a project that succeeded only after this shift occurred in mid-course.

Due to the growing demand for more usable systems, practitioners may find a climate for limited experimentation with these and other approaches. But even to *begin* working effectively requires a clear awareness of the obstacles, an understanding of why they are there, and a tolerant recognition that their source is in institutional constructs, not in unsympathetic individuals.

PART II. WITHOUT KNOWLEDGE OF USERS: INTERFACE DESIGN BY DEFAULT

Everyone in an organization is working toward many goals at any given time. These include personal goals, team or group objectives, and the purposes of the organization as a whole. Goals arising from community concerns, professional group affiliations, and elsewhere are also introduced into the work environment. There are far more goals in an organization than there are people -- which for a large development company means many goals indeed!

These goals do not all conflict, of course. Companies strive to see that employees share the corporate goals. Teams are established in part to foster common goals of a more focused nature. Ideally, people are working in concert or toward goals that have only incidental affect on the work of others. But of course, goals often do conflict. No one gets everything they would like. Adjustments happen. There is an ongoing process of discovering conflicts, arguing, compromising, horse-trading, adjusting, re-prioritizing, and accepting. Much of it goes on unnoticed -- we have highly developed skills that allow us to work unconsciously toward many of our objectives.

Thus, in a product development organization, even someone whose job is limited to interface development is addressing many goals simultaneously, just one of which is developing a good interface. For those whose professional responsibilities extend beyond the interface to include system design, programming, marketing, managing, etc., there is even greater parallel goal-driven activity. Other common goals include designing and implementing reliable software and getting products out on time and within budget, obviously. Equally evident but of a different nature are such goals as communicating design ideas, initiating promising projects, and assigning people where resources are needed. Goals such as insuring that people feel good about their work and are rewarded appropriately may seem tangential to interface design. However, the web of activity in an organization is tightly woven, and as the hundreds of people in an organization work skillfully, often unconsciously, toward thousands of goals, a decision made in one area to address one concern will inevitably have effects, however subtle, elsewhere. Indeed, steps taken to reach virtually any goal *could* influence an interface design. These indirect effects may not be intended or even noticed -- yet they can systematically distort the interface, introducing elements that work against the eventual users of the product. This is particularly likely in the absence of knowledge about what the users might prefer. Confronted with a set of interface design alternatives and lacking knowledge of what would actually best serve users, a team may find that one of the alternatives helps them attain another goal as well. For example, it may be more easily described and communicated. This part of the paper describes possible undesirable influences exerted on the interface by goals that are tangential to interface design. Developers should watch for these conflicting forces, which often go unnoticed, and persist in efforts to

learn about users and their work environments. The competing goals are often themselves genuinely important, so a solid case may be needed to convince colleagues to compromise them on behalf of the users.

Software Development Goals

Individual Goals

Understanding the software architecture
Design simplicity
Design consistency
Anticipating low-frequency events
Thoroughness

Retaining responsibilities
Extending skill repertoire
Personal expression

Group or Team Goals

Communication
Coordination
Compensation
Cooperation

Organizational Goals

Division of labor
Efficient decision-making
Organizational survival

Figure 2. Goals that may conflict with interface design.

Figure 2 categorizes the goals to be discussed into individual objectives, group or team objectives, and organizational objectives. This is similar to the "layered behavioral model" used by Curtis, Krasner, and Iscoe (1988), with the principal exception being that they used the categories of "team" and "project," collapsed here into one category that is further extended to include groups that cut across project boundaries, such as task forces. The attribution of goals to any of these entities is at best imprecise. An individual may internalize group or organizational objectives. It can be difficult to pin down the objectives of groups, the often amorphous and overlapping organizational units wherein individual and organizational aspirations merge. Nor is complete consensus likely regarding corporate goals. Nevertheless, the existence of personal objectives is evident, and one finds conscious efforts to create solidarity around specific team and organizational objectives. Another complication is that the same behavior often serves multiple goals, including goals operating at different levels. For example, the goal of "design simplicity" may reflect either a designer's personal aesthetic values or a contribution to the team's engineering process, or both. Nor is this list

comprehensive. The description that follows should therefore be considered suggestive of conflicts that can occur, intended to portray some of the complexity of the development environment. The hope is that readers will be vigilant lest these or other goals operate quietly to distort the interface design process.

SOFTWARE DEVELOPMENT GOALS

Certain constraints that may affect the interface are particularly familiar because they force tradeoffs or compromises in other aspects of software development. These competing goals are often the greatest impediment to usability. But most developers are all too aware of them, and stressing the familiar could distract attention from the subtler problems that more often escape notice. So while we will just briefly mention a few in this section, the lack of attention by no means suggests relative lack of importance.

The goal of meeting tight schedules. Product developers are usually under pressure to produce new releases and new products quickly. In a survey of over 200 professionals in six disciplines, each with some role in interface design, Grudin and Poltrock (1989) found that "insufficient development time" was estimated to cause "substantial impairment" to over one-third of all the interfaces developed -- the greatest percentage attributed to any one factor.

The goals of minimizing memory and processor use. The rapid drop in cost of computer memory and processing time make better interfaces possible, and create a demand for better interfaces by making computation accessible to more people. Nevertheless, minimizing the use of these resources is still a major goal on most projects. This creates pressure to constrain the size of on-line help or reference texts and to limit processor-intensive interface activities. For example, research systems that explore advanced interface modalities, such as video, and advanced techniques, such as user modeling and coaching, utilize resources that are still out of the range of most products. The human factors engineers responding to the aforementioned survey reported "implementation constraints" as causing the most substantially impaired interfaces (38%). ("Implementation constraints" was not further decomposed; it could apply to constraints other than memory and processing time, such as those below.)

The goals of producing reliable, maintainable, secure software. Adding complexity to create interfaces for people with differing roles, experience, and preferences comes with a price. Security considerations may require computer users to carry out more steps than they would otherwise.

These and other goals integral to software development can compete with usability. They are all, of course, valid goals in their own right. In the absence of information about what computer users really require, it will be very difficult to justify interface design approaches or choices that compromise the efforts to reach these goals.

COGNITIVE PROCESSES AND INDIVIDUAL GOALS

System development gives rise to many goals that operate primarily at the level of the individual developer. There is not always a sharp distinction between an individual goal and a team or organizational goal. In addition to goals that originate in system development responsibilities, individuals have personal goals arising from career plans, preferences, style, even personality. Conflicts based on these can emerge in the turbulent world of software development in general and interface development in particular. Again, influences from different levels flow together. Personal attitudes shape one's interactions with members of support groups -- and the organizational context also plays a major role.

Cognitive activity lies at the center of much design and development work, whether it takes place at a terminal, with pen and paper, or in group settings. System development is a highly rational activity. Of course, other processes are involved -- perceptual and motor at the terminal; motivational, emotional, and social in group contexts; organizational in still broader contexts. Many of these are discussed in this and subsequent sections. In considering individual goals in the workplace, however, cognitive processes are central. Behavior is governed by cognitive abilities and limitations interacting with tasks and objectives. The manifold goals present in the development environment act in conjunction through cognitive tendencies to influence interface design. A few of these cognitive tendencies are:

Our inability to forget! Most of the time, we complain about our bad memories, but sometimes it would be convenient to be able to forget on command. If we could put out of mind what we know about how a

computer works or how we ourselves use it, perhaps we could see how someone less knowledgeable or with a different job will experience it. But this we cannot do.

Selective memory for salient events. Our memory is not uniform, however. We remember some things better than others. This can interfere with objective design -- we exaggerate the importance or frequency of the things that we attend to and ignore the frequency or detail of unexceptional activity that takes place.

Difficulty making conscious the unconscious. In fact, we are not even aware of most activity -- our own and that around us. We are so highly skilled and practiced at acting to reach many of our goals that doing so does not require conscious reflection. The resulting lack of awareness makes it difficult to detect or deduce the true causes of many design decisions.

Consciously handling multiple "channels." Although practice does help, we are not good at consciously working out the interactions of complex, multi-layered situations. Since this describes most work situations (for example, the software product development environment as outlined here), we are not well equipped to understand them. This has obvious implications for developing computer support for groups. With practice we may learn to handle such complexity unconsciously, but in thus delegating the process to uninspectable intuition, we allow other highly practiced activities to become part of the basis for our actions. The effects on design of these "other highly practiced activities" and the non-design goals that they serve are explored in this part of the paper.

Individual Goals Arising From Professional Responsibilities

The Goal of Understanding The Software Architecture

Apart from carelessness, the most common source of poor interface design may be the natural tendency of developers to "map" elements of the underlying software architecture onto the interface. When the interface reflects the underlying design, only one "model" of the system must be kept in mind. In the early days of a new technology, most users are technically proficient, and it may be useful *as well as* convenient to reflect the architectural or "engineering model" in the interface. As the user population becomes less technical, the need is recognized for different interfaces. The engineer is aware of the relationships among

functions based on the implementation, whereas users construct different relationships among the same functions based on the roles they play in the users' tasks.

Gentner and Grudin (1990) describe several cases in which the "engineering model" of a system was imposed on the user interface, to its detriment. In one, a commercially available VCR remote control labelled its buttons 0-9 and A-F, corresponding to an underlying hexadecimal representation but confusing to most users. In another, a software design team resisted adding a new function to a menu even though users expected it to be there, because the function was implemented quite differently from the other functions on that menu. Similarly, developers decided the "print" function should handle all system objects consistently according to their internal structure, even when this led to printing a folder (or directory) index when users expected to have the objects in the folder printed (Grudin, 1989).

Even when intellectually recognizing the need to separate the software design from the interface design, an engineer cannot willfully "forget" the architectural model and assume the users' perspective. This point was emphasized by Gould and Lewis (1985) and illustrated metaphorically by Landauer (1988) with a "hidden figure" in a photograph: it is difficult to see the figure, but once it has been pointed out, you will always see it. You cannot return to the naive state.

The Goal of Design Simplicity

In a review of work on "programming aesthetics," Leventhal (1988) explored what it is about certain programs that appeals to programmers. Among the factors frequently mentioned was the notion of simplicity. "Programs have a kind of simplicity and symmetry if they're just right," said one programmer. The author summarized that "themes of simplicity and wholeness, as well as the taming of complexity are suggested."

Simplicity serves other goals as well. A simple design is likely to be easier to communicate to other team members, to test and to modify. For all of these reasons, a simple, elegant technical design is highly prized.

One might assume that computer users would also find a simple interface easier to use. This is not always true. When there is a distinction that people already make or would benefit from making, designers who "tame complexity" and create a simpler interface by eliminating that distinction

are creating an inferior interface. The potential value of introducing complexity by creating a distinction not required by the system can be seen in a desktop metaphor interface sitting on a Unix operating system. Unix files are represented by "document icons" and Unix directories are represented by "folder icons." Simple enough, so far. But complexity has been added: the interface also provides "cabinet icons" and "cabinet drawer icons." Like the folder icons, these correspond to Unix directories. A suffix in the name determines the icon to be displayed. Apart from that, the directories are indistinguishable. Why complicate things by having 3 different representations for directories? And the directories *are* treated uniformly -- for example, one can place a cabinet inside a folder just as easily as the other way around. The complexity is introduced to provide a methodical way to organize directories hierarchically. Strictly to help users manage their files, the designers added a distinction not required by the system.

In this case, the designers wisely overcame the attraction of simplicity. This doesn't always happen. Grudin (1986) describes a similar desktop system in which under the surface, all documents are located in a single central "catalog." For users to see a document displayed in a directory (appearing as a folder), the system places an internal pointer corresponding to the address of the document in that directory. A document can be "shared," or appear in two directories, when identical pointers are placed in each directory. From the system perspective, the two directories always share the document on an equal footing -- there is no way to treat them differently, because the pointers are indistinguishable. However, users may *expect* the "original" document -- the one appearing in the folder where it was first created -- to be treated differently than the "linked" version created later. For example, when a "remove" command is issued, the dialogue might be different if the original is being removed than if the subsequent linked copy is being removed. This could be handled by designing *more complex* pointers to reflect information about object creation. But the developers preferred to stick with the *simple*, elegant pointer implementation even when it was observed that it provides no way for the interface dialogue to accommodate the users' expectations.

Shneiderman (1987) quotes Nelson on the goal of simplicity: "Designing an object to be simple and clear takes at least twice as long as the usual way. It requires concentration at the outset on how a clear and simple

system would work... It also requires relentless pursuit of that simplicity even when obstacles appear which would seem to stand in the way of that simplicity." Sometimes, one of those obstacles to simplicity is good interface design.

The Goal of Design Consistency

"Experimental results... show that consistency (leads) to large positive transfer effects, that is, reductions in training time ranging from 100% to 300%," (Polson, 1988). "Build consistent user interfaces," (Rubinstein and Hersh, 1984). "Strive for consistency. This principle is the most frequently violated one, and yet the easiest one to repair and avoid (violating)," (Shneiderman, 1987). "The common application of design rules by all designers working on a system should result in a more consistent user interface design. And the single objective on which experts agree is design consistency," (Smith and Mosier, 1986).

Developers have indeed taken the goal of consistency to heart, but unfortunately, consistency can work *against* good interface design in several ways. As noted above, an interface that is consistent with the underlying software architecture may not be ideal for the computer user. It is often sub-optimal to consistently use engineering terms to label objects (e.g., the "BREAK" key), to consistently reflect an "engineering perspective" in error or status messages (e.g., "read error" when "check for disk damage" might be more helpful), to group objects in menus according to aspects of their implementation, and so forth. A second problem is that consistency with a real-world analog that facilitates ease of learning may obstruct ease of use. Several examples appear in Grudin (1989). For example, several calculators initially adopted alphabetically arranged keyboards. This arrangement may help in some initial learning situations due to its consistency with our experience of the alphabet, but it slows down experienced users. Another problem with the goal of consistency is that of choosing the dimensions on which to be consistent. There are often many conflicting possibilities -- each consistent, but not all of them good. Everyone recognizes "a foolish consistency," such as consistently capitalizing every other letter in command names, but there are subtler, less obvious cases. For example, in abbreviating command names, truncation ("de" for delete) is better when users will know the name and have to recall and type the abbreviation, whereas "vowel deletion" ("dlt" for delete) may be better when the users will see the

abbreviation and have to recall the name, as on a keycap, and single-letters ("d" for delete) may be better for highly over-learned or prompted commands where typing economy is critical, as in email options. A designer could create a consistent set of abbreviations, but if it is mismatched to the users' tasks, it will be a poor design.

While some form of consistency may be helpful, especially when we are relatively ignorant about the users' work practices, an optimal design may not be "consistent" in any obvious way at all. Considerations such as dialogue efficiency or the prevention of damaging accidents may outweigh consistency in importance in a given situation (Norman and Grudin, 1990). Only by understanding the users' situation can developers get a handle on this. Consider the task of categorizing wildflowers. Botanists sort on the basis of leaf and petal shape, and many nature guides reflect this organization. But amateur nature lovers may start from the color of the flower -- and some wildflower books organize flowers in sections by color. Insects may categorize flowers by smell, and deer by taste. Since they can't read, we don't find books organized in these ways, but the example illustrates the diversity of possible approaches to objects. Consistency chosen with the wrong task in mind can work against good design.

The Goal of Anticipating Low-Frequency Events

A system or application design must cover all contingencies. Because extremely low-frequency situations are precisely those that may escape diagnostic testing, developers think carefully about possible combinations of conditions and events. Discovering a potential problem through logical analysis is a potent source of peer approval. For the same reason, diagnostic testing itself focuses attention on low-frequency events.

This concern with complex contingencies, coupled with our tendency to exaggerate the salience of the things we focus on, may give extreme cases a disproportionate influence on the design. Interfaces may foreground features for circumventing or recovering from rare problems, cluttering an interface or distracting users. Alternatively, a potentially useful feature that will break down in exceedingly rare situations may be abandoned early in the design process, when it could have been retained with suitable notifications or safeguards. Preoccupied with logically possible contingencies that can reach "Rube Goldberg" complexity, we forget how rarely they will actually occur. The engineering maxim "if

something can go wrong, it will," has unquestionable validity, but must be kept in perspective. I have watched many babies leave with the bath water. For example, while a particularly sensitive information system may have to adopt security procedures that burden users, it may not be desirable to similarly complicate the interface of a system for use in environments where mischief is highly unlikely.

Some interfaces do a good job of moving rarely used features to low-level menus or bringing them out in appropriately timed messages or prompts, rather than forcing them on our attention. However, this aspect of design must be monitored carefully, in part because several other goals described below also lead to undue emphasis on low frequency operations.

The Goal of Thoroughness

Every software procedure goes through a similar process of being described, written, reviewed, and documented. There may be a standard documentation practice: one procedure per page. These and other "checklist" activities reenforce a perspective in which all functions are of uniform significance. This goal of being methodical and thorough, while an important part of the development process, disguises the tremendous variability in importance and frequency of use of different operations *to users*. The goal of thoroughness thus conceals the frequency and importance of some features in the computer use environment, just as does the goal of anticipating low-frequency events.

This can distort the interface in several ways. The "Help" function on some systems lists dozens of commands in alphabetical order, leaving the user to go through sifting out obscure and similar-sounding options. Similarly printed documentation may simply list functions alphabetically or one-per-page in a manner directly reminiscent of the developer documentation on which it may be based. Of course, if one knows nothing whatsoever of the computer use environment, this approach may be the best one can do. But by obtaining *any* information about that environment one should be able to arrange things more usefully.

This same perspective has brought into question the utility of considerable experimental work in the field of human-computer communication. Many studies of menu arrangement have examined menu depth, breadth, and other variables. In these studies the different possible menu choices are tested uniformly. Yet we may be confident that menu choices are not

selected even remotely uniformly in work settings; frequency is perhaps the most significant variable and one would expect frequency variations to outweigh other factors in menu arrangement.

Individual Goals Arising From Personal and Career Issues

The Goal of Protecting Turf

Specialization in software design is a relatively recent but rapidly spreading phenomenon in product development companies. In the past, programmers and software engineers developing interactive software typically designed the user interface along with the other components. Many still do, but the growing importance of improving interfaces creates a role for interface specialists. These include human factors engineers, technical writers, graphic designers, and others. Specialization will increase as new media -- color, sound, video, animation -- are more widely incorporated. Some have called for the involvement of specialists in the dramatic arts (e.g. McKendree and Mateer, 1989; Mountford, 1989).

This new division of responsibility may be resisted by engineers who are losing part of their traditional responsibility (e.g., Blomberg, 1986). Some engineers are happy to be relieved of this uncertain part of the design, but others see the interface as an enjoyable challenge or are acutely aware that users will judge the software on the basis of the interface, the visible part of the product. Perhaps because programmers feel particularly competent at technical writing (Grudin and Poltrock, 1989), development teams often exclude writers (e.g., Good, 1985; Gould et al., 1987).[4]

---

[4] There is an irony in these two studies. In Good (1985), technical writers handled the printed documentation, while a software engineer designed the help and error message text. Of the 362 suggestions received during iterative development, the most frequent category was "suggestions for improving the wording of particular help or error messages that users found to be confusing." In Gould et al., (1987), engineers retained control of the user guide, and "we iterated over 200 times on the English version of the user guide." Each case study is presented as a demonstration of iterative design. Iteration is unquestionably extremely valuable, yet one has to ask whether *such extensive* iteration would have been required had professional writers designed the help messages, error messages, and user guide.

The Goal of Staying Current

The expanding set of interface elements and modalities produces a related effect. Developers may find it interesting to work in these new areas or may feel that they must lest they find themselves with obsolete skills and declining responsibilities. Engineers with no prior experience in graphic design, assigned to a project centered on a bit-mapped display, may enjoy designing icons, window borders, etc. An engineer whose graphic designs are acclaimed within the programming group may resist working with (or yielding to) a real graphic artist whose expertise would improve the product.

* * *

The goals of retaining and extending one's responsibilities, while potentially undermining interface quality, are consistent with one of Gould's (1988) four key principles for designing usable systems: keep responsibility for usability under one roof. When one programmer has total responsibility, this condition is met. While that is not Gould's intent, it nevertheless points to a fundamental organizational challenge presented by the explosion of specialization -- how to coordinate the specialists effectively. This problem recurs in the discussion of goals at the group and organizational levels.

The Goal of Personal Expression

An important if largely unconscious goal for each of us is to achieve some psychological balance, whether by obtaining attention, approval, respect, power, or whatever. This seems uncontroversial, yet it can be difficult to discuss in the context of engineering, which has the antithetical goal of highly rational behavior. Nevertheless, psychological factors are critical to the success of development projects (e.g., Shneiderman, 1980; Boehm, 1988). Actions taken in furtherance of these personal goals have clear influences on the course of development, including interface development. The "Not Invented Here Syndrome" is an example.

One writer has pinpointed the potential for conflict in an outspoken manner: "Historical accident has kept programmers in control of a field in which most of them have no aptitude: the artistic integration of the mechanisms they work with (in interface design). It is nice that engineers and programmers and software executives have found a new form of

creativity in which to find a sense of personal fulfillment. It is just unfortunate that they have to inflict the results on users (Nelson, 1990)."

* * *

Actions undertaken to satisfy personal and career goals can undermine the cooperation between software engineers and interface specialists in support roles: human factors, technical writing, industrial design, training, marketing, etc. Productive communication and cooperation may be further diminished by the lack of the experience companies have coordinating this relatively new activity. Differences in outlook and "language" may inhibit mutual education and trust. Development groups sometimes try to circumvent these problems by absorbing the support functions or establishing small support groups internal to the project. This may work for a time, but when a project is completed and the product is turned over to others for revision or maintenance, responsibility for the support materials (documentation, training, etc.) tends to revert to the external support groups established for this purpose by the organization. Aware of its initial exclusion, a support group may cheerfully throw away the version developed by the team and start over. The resulting confusion is unlikely to benefit users.

SOCIAL PROCESSES AND GROUP OR TEAM GOALS

Group dynamics introduce a new set of social, motivational, emotional, and political concerns that are less apparent when focusing on the work individuals do in isolation. Goals at the group or team level are often unstated, possibly even unrecognized, and thus their influences may be less apparent. Communication and coordination are two key activities that are largely cognitive and amenable to rational analysis, while goals such as equitable treatment and cooperation introduce more explicit motivational and emotional elements.

The Goal of Communication

An interface design that is easily communicated is helpful in the development environment. Unfortunately, the resulting interface may not be so easy to use when it appears in the user environment.

Few interface designs in large development organizations are written, reviewed, approved, and implemented by one person. As a result, a design must be communicated. Communication of all sorts is a

fundamental activity in such organizations -- skills may vary, but they are heavily exercised. For communicating a design, paper is the preferred medium and brevity a preferred style.

Some designs are easy to describe on paper and some are messy. A set of pull-down menus no more than two levels deep, each menu with about ten items, looks elegant on the printed page. A design with branch points extending in places three or four levels deep, some areas bristling with options and others sparsely populated, may be a real mess on paper. In the absence of hard evidence as to which users will prefer, why not go with the easily communicated design? It even seems to make sense that a design that is easy to describe will be easy for a user.

But it is often not true. A user's dialogue with the computer is narrowly focused and extends over time, in contrast to a static, spatially distributed written design. Presenting options at different points as a user proceeds may make good sense, however confusing it appears on paper. Menus that vary significantly in length and depth, depending on the contents, may work well, even if inelegant on paper.

It is also easy to overlook interface problems on paper that uses will stumble over repeatedly. For example, it is easy to display on one page an entire set of pull-down menus. Readers searching for a particular item can always find it in seconds, even if it is not under the first heading that they consider. But users never see the entire set of menus simultaneously. If an item is under the wrong heading, they will wander around, inspecting possible synonyms or dropping down to search lower menu levels fruitlessly. They may encounter the same difficulty several times before they finally learn the location of the item.

Similarly, designers may be discouraged from placing an option in two places in the interface. It looks inelegant on paper and he gratitude felt by a user wandering through the avenues of the interface is difficult to imagine. (The tendency to map the software architecture onto the interface and the "logic of the checklist" noted under the goal of thoroughness also work against including multiple paths to a single operation. Procedures and their definitions are not placed in multiple places in the code or the code documentation.)

Problems based on the goal of communication are very hard to detect on paper, yet may be uncovered quickly with even primitive prototyping, especially if the prototypes are tested in conditions approximating work

settings. This underlies recommendations that prototypes replace documentation for the communication of design proposals.[5]

The Goal of Coordination

Fairley (1985) observed that the software architecture of large systems has tended to reflect the organizational structure of the development organization producing them. In the same way, an interface can reflect the structure of the group that is responsible for developing it.

We discussed the developer who imposes the "engineering model of the system" or the underlying software architecture on the interface, perhaps the most prevalent systematic source of interface problems. The benefit in doing this is that the engineer can work with a single model covering both the system and the interface. The "management model of the project" presents an analogous case. Managers must assign implementers to functions. They also encounter these functions in reviewing the interface design. In fact, if the team follows the recommended practice of writing "the user manual" first, the manager deals with both simultaneously. It greatly simplifies things when the models coincide: when the interface reflects the assignment of programming tasks.

If a manager who needs to obtain an additional programmer to code a specific function may find it easier to make the case if the function is called a "utility" (and ultimately appears on the "Utilities" menu) than if it is described as being part of another function that is already being coded by someone else. However, the user might expect to find the two functions together, rather than having to leave one to explore the Utilities menu. Similarly, when one implementer handles several functions, they may gravitate to the same interface location.

Default design decisions such as these, made for the sake of convenience, may yield quite easily to clear evidence that they cause users problems. But that evidence must be provided.

---

[5] Even superficial aspects of communicating on paper can influence design decisions. A "professional-looking" design document begins with a real advantage. I worked in a development group in the mid-1980s where the Macintosh quickly became an indispensable design tool due to the appeal of documents incorporating its graphics.

The Goal of Compensation

When a designer has created a novel feature or when a programmer has completed a difficult implementation, it is desirable to find ways to reward the effort. One approach is to increase the work's visibility through prominent placement in the interface. Of course, from a user's perspective, the appropriate time and place to provide access to a feature may be buried in an interface that gives prominent placement to more ordinary features. The result is a new command defined when adding a parameter to an existing command would suffice, a new function key assigned in when it would be better to extend the use of an existing function key, and so forth.

Solid evidence of user work patterns or preferences may be needed to justify a decision to "bury" a feature in an interface. A friend once joked, "I work on this two years and you get to it by hitting CONTROL X in some other application?" The goal of compensation may work in concert with other goals to promote visibility at the expense of usability. Visibility may serve marketing purposes: it may appeal to *buyers* or *customers*, not users. (Sometimes a customer subsequently becomes a user, sometimes the customer acts on behalf of users and may focus exclusively on functionality.) Or its placement may reflect its prominence in the developers' minds when it is completed.

Consider this example: Some stand-alone menu-driven office systems developed in the early 1980s include a spelling checker that appears high in the menu hierarchy. To use it, one closes the document being edited, moves up in the menu hierarchy, selects the spelling checker, then reenters the name of the document that was just closed. Following spelling correction, to resume editing one accesses the word processor and reenters the document name. This is extremely inconvenient: one would generally prefer to enter the spelling checker from within the word processor (although perhaps not by typing CONTROL X). We do not know which factors were present -- ease of implementation, visibility for marketing purposes, the perceived importance of the feature, the need to coordinate or compensate developers -- but we *can* be sure that the decision was governed by goals other than usability![6]

---

[6] In Lewis and Polson (1990), this interface is examined in studying a program that formally analyzes interfaces for usability problems. The program does detect such problems as a syntax problem in the command parameters, it did not catch the menu

The Goal of Cooperation

Compromise and finding workable tradeoffs is characteristic of most aspects of life in development environments. To improve the interface requires sacrifices elsewhere. It would be easier if the interface reflected the underlying software architecture. It would be easier if the interface reflected the organization of the code documentation. It would be easier if formal properties of consistency or simplicity could be applied to interface design. Where is the tradeoff point, when do we compromise? In the absence of strong evidence of usability problems, being cooperative may mean compromising the interface uncomfortably often.

The give and take present in the engineering environment should not be underestimated. All does not work by purely rational decision-making: People "call in IOUs" and "cash in a lot of chips" to reach important goals. As mentioned in Part 1, the lack of feedback from existing users robs the interface of something that might give it additional weight: the awareness that it affects a lot of people for a long time. Users are still working with it long after the developers have forgotten it. Because developers can quickly put it out of mind, the interface is easily compromised. Even when the users' perspective *is* in focus, it can fall victim to good-humored compromise:

Several years ago, developers from two of a company's merging product lines met to unify their "look and feel." One product used a 24-line display and status messages appeared on the first line. The other product used a 25-line display and messages appeared on line 25, at the bottom. Since future customers would use both products together, consistency was desirable. No agreement was reached following hours of debate over the merits of each and the difficulty of changing either. Finally, a solution was proposed: Both products would consistently agree to display prompts consistently -- on line 25. On the 24-line display, this would of course wrap around to the first line. The proposal was accepted.

To avoid unduly sacrificing the interface in the spirit of cooperation, we need the voice of the user. Developers have some difficulty arguing with users -- we have all heard the expression "the customer is always right."

---

organization problem. To recognize it requires an understanding of the users' work environment that is beyond the grasp of such systems. While it may add to a developer's inventory of tools, formal interface analysis will not replace the need for user involvement.

An oversimplification, granted. But we *never* hear "the user interface specialist is always right." So a user is a good ally to have.

ORGANIZATIONAL PROCESSES AND CORPORATE GOALS

Organizations can be considered as units, interacting with other organizations, reacting to external events, and developing internal structures and processes to help with these activities. Organizations prefer to stay in business, to survive, and may concern themselves with developing a "corporate culture" that will carry them through changes. Mintzberg (1984) argues eloquently that organizations have a great latitude in both creating and reacting to their environments. He further suggests that only certain combinations of internal and external conditions lead to organizational stability, which if true means that successful organizations should strive to reach, maintain, or shift among these stable arrangements -- which means adopting specific sets of organizational goals. This section explores a few basic organizational goals and their influence on interface design.

The Goal of Division of Labor

Part I focused on the effect of division of labor in separating developers and computer users. Contact with customers and users is channeled into Sales, Marketing, Field Service, Customer Support, Training, and Management. An example of secondary, intra-organizational effects was the case of on-line help, its usefulness recognized in Customer Support but not in Development.

Figure 1 shows the assignment of responsibility for different parts of the interface to groups in very different parts of the organization, a further source of fragmentation and communication challenges. Marketing defines high-level functionality, software and human factors engineers in Development design the low-level software dialogue, Technical Publications writes the documentation, and Training develops on-line and training courses. Communication is often primarily in the form of written specifications (between Marketing and Development), functional documentation (supplied to Technical Publications by Development), and the actual software (provided to Training). The possibilities for miscommunication are legion.

Poltrock (1989c) provides a nice illustration of how usability can be affected. Developers in a large company produced a new release of a

popular product, introducing improved methods for accessing certain functions. The developers planned to phase out the original, less efficient interface, but retained it as an alternative in this version to allow backward compatibility for current users. Training Development, unaware of the overall picture, developed training materials that taught only the old, inferior method.

Specialists in organizational structure will have to determine how large product development organizations can address this problem. The current organizational structures worked more efficiently for the development of non-interactive systems. The increased demands for communicating user needs have been handled by beefing up the use of internal and external mediators: marketing and system analysts, consultants, etc. The many voices calling for direct user involvement in design are explicitly questioning the soundness of this approach. Perhaps large organizations can take steps to increase user involvement without much difficulty, or perhaps major changes will be needed. This issue is returned to in the conclusion.

The Goal of Efficient Decision-Making

Decision-making involves a tradeoff between two desirable goals: maximizing confidence in the correctness of a decision and reaching that decision in a timely, efficient manner. Inevitably, intuition plays a large role in most corporate decision-making. Executives and managers in development organizations generally have good track records, but interface development is a new concern. Decision-makers have not had the experience to build or demonstrate their intuitions in this area.

If anything, decision-makers in the development environment often support questionable interface technologies, such as voice and natural language (Grudin, 1988; 1990a). Managers who do not use computers think that perhaps if learning time were reduced to zero, they would use a computer after all.[7] These technologies have remarkably long histories of failure and exaggerated forecasts (e.g., Aucella, 1987; Johnson, 1985). The problem is two-fold: i) the technologies themselves are extremely difficult to perfect (or even render adequate) -- projects in these areas

---

[7] Actually, many managers would like computers to be like people, with whom they interact skillfully. More explicit than most, Nicholas Negroponte, Director of the MIT Media Laboratory, says "computers should be more like people," (1990a) and "the computer must be an old friend," (1990b).

have been characterized as "black holes" (Williams, 1990); ii) when available in circumscribed form, the technologies appeal at best to users in restricted niches -- for people who *do* use computers heavily, which does not include most decision-makers, the drawbacks outweigh any advantages that the current state of the art can provide.[8]

The failure of intuition is a particular problem for the development of "groupware," products designed to support groups. A manager with good intuition may be able to use a spreadsheet for an hour and decide that many users will like it, but no one person's intuitions will cover the range of needs of group members who differ in role, experience, and preferences. The complex social dynamics into which such an application will be injected are difficult to forecast. Not surprisingly, development managers tend to favor applications that would benefit managers, not realizing that the work required of other users may preclude their success. Thus, considerable effort has gone into project or work management applications, decision support systems, meeting scheduling and management systems, voice applications, etc. -- despite the fact that the more successful groupware applications, such as electronic mail, code management systems, and databases, are those that do *not* selectively benefit managers.[9]

The Goal of Organizational Survival

In addition to the inward-looking concerns described above, large product development organizations must manage their external relationships with customers, competitors, and others. These concerns create powerful forces that often intrude directly into the design process. One such force, discussed in Part 1, is the pressure for rapid releases that can work

---

8 This problem has an impressive pedigree. Licklider and Clark (1962) included speech recognition and natural language understanding "including syntactic, semantic, and pragmatic aspects" among ten prerequisites for true human-computer "symbiosis," although they seemed more aware of the difficulty than many who followed. In "Information technologies for the 1990s," Straub and Wetherbe (1989) forecast that "human interface technologies" will be the information technologies with the greatest impact, and speech recognition and natural language will be the key human interface technologies. Not surprisingly, the article was based on interviews with seven corporate presidents or chairmen, two executive directors, and one business school professor. Equally optimistic is Negroponte's August, 1989 prediction that "the most significant development in the human/computer interface during the next five years will be in speech technology," (Byte, 1989).

9 Disagreement exists over what is properly classified as groupware. For the present purpose this is not important. This issue is discussed in Grudin (1990c).

against user involvement. Other organizational goals and their effects include:

A competitor may introduce a new feature and obtain a marketing edge by promoting it, whether or not the feature provides any real benefit for users. In response, a company may develop the same feature without ascertaining or even caring about its contribution to usability. This process may explain the spread of some unused software features.

In a rapidly-changing marketplace, the existing users, "the installed base," is both a blessing and a curse. It provides a relatively reliable market for new products, yet exerts a conservative pressure against change. Interface change in particular requires existing users to adjust. Most change incurs a cost. Overcoming resistance to an interface change may be easier with empirical data concerning the degree of disruption and eventual productivity gains it will cause -- data that are rarely collected. Yet, change must be introduced: the last remaining users of paper tape or punch cards may like their familiar technology, but a product company cannot be held back too long.

What is worse, even when one can demonstrate that most or all users will benefit from change, organizational survival may stand in the way of adopting the improvement. Even worse than having a less-than-state-of-the-art product may be having a reputation for abandoning existing customers. However small or irrational the group opposing change, the product development company wants to avoid being known for deserting its users. Thus, outmoded features may clutter interfaces and development resources may be channeled into extending their lives, at the expense of innovation and usability for future users.

CONCLUSION: MULTIPLE CONSTRAINT SATISFACTION

The software development process is one of constant compromise. Tradeoffs may be forced by conflicting engineering considerations, management decisions, and product marketing constraints. Development time may be pitted against additional functionality, hardware capability against price considerations. Tradeoffs also emerge from the other forces described: one design may be easier to communicate than other, one group may deserve greater recognition, two projects may be competing for one available engineer, and so forth. Designing and building a quality interface is always a goal, but it is just one in many. In the absence of a

strong case for particular choices, aspects of the interface may be severely undervalued when the tradeoffs must be resolved. Interface quality may be compromised much too readily.

In Part 1, the obstacles that product developers face in involving user in design and evaluation were described. Part 2 motivated efforts to overcome these obstacles by describing forces that may systematically distort interfaces. The best way to overcome these forces -- or more accurately, to balance them, since most of them represent legitimate goals -- is to obtain clear evidence of what actually is required to enhance usability. In the struggle to satisfy multiple constraints, silent, weak, or indecisive users' voices do not represent a strong constraint.

A second purpose of this paper is to hint at the complexity of workplaces, to indicate what will be involved in understanding work environments well enough to position computers in them. The workplaces described here are special -- product development environments -- but it is safe to assume that many or most computer *use* environments will be equally complex. Computer use environments are increasingly different than computer development environments. Developing sufficient understanding of their complexity will not be easy.

Product development companies can address the problem of increasing developers' understanding of users in several ways. They can redefine their users, they can strengthen their use of mediators, they can reorganize or restructure their development methods to increase direct contact, and they can use technology to increase the flow of information.

Redefining the User Population

As computer use extends to more application domains and as product maturity in those domains increases, large product development companies have to decide for which users they will develop. They may focus on specific markets or may channel their efforts to third-party developers and value-added resellers with experience developing software for specific domains. Both of these trends were noted by the president of the Dutch electronics company Philips in announcing plans to narrow its focus to specific market segments, such as banking. "There is a shift from selling directly to customers and retailers to selling indirectly through value-added concerns who tailor the products to the customer needs," (International Herald Tribune, 1990). This, in turn, means

adopting industry-standard platforms. In taking this path, software development follows hardware, which ceased being sold to "end users" some time ago.

This does not fundamentally change the issues. Product development companies still must understand their users, but since their users are themselves software developers, the gap that must be bridged is narrower. Responsibility for understanding the ultimate users of the system moves to the value-added reseller, operating in a narrower domain. As these companies grow and compete, they will face some of the same issues that now face the large product development organizations.

Strengthening the Use of Mediators

As interfaces continue to grow in importance and as experience with them increases, the traditional mediators or indirect communication paths between developers and users will do a better job. Systems analysts, marketing personnel, consultants, user groups, and others will become more skilled at identifying and describing interface needs. Standards organizations will devote more attention to interface issues. Software engineers, human factors engineers, and other developers will become more knowledgeable about interfaces and interface development methods. How successful these approaches can be, operating alone, is not clear. The complexity of work environments suggests that they may always be marginal at best.

Organizational Change

Traditional software methods are under considerable pressure due to the recognition that they are inadequate for developing interactive systems (e.g., Boehm, 1988). Methods such as prototyping and iterative development are widely accepted as necessary. Many design faults that arise due to tangential forces can be relatively easily filtered out with such techniques. Yet we have identified a number of ways in which current organizational structures and practices work against the application of these methods. What is more, their application requires some level of active user involvement. Bringing this about will require organizational change, beginning with a change in awareness.

Technology Support

Rapid prototyping tools are a standard but critical recommendation. Prototypes may come to replace written documentation for interface specification. Prototyping tools could be integrated into the development system or facilitate cooperation among computer users and developers (see Grønbæk, 1990). Just as code management systems have proven useful in multi-programmer environments, so single-user rapid-prototyping tools may benefit by adopting features designed to support the collaborative nature of most development.

Video and other multi-media software support may enable developers to communicate more effectively across distances. More importantly, video may allow much greater communication of aspects of users' experiences to developers. Video can be remarkably effective at conveying the specific details as well as the general richness of work environments.

The need to bridge the information gap between development and use environments is so great that the computer should come to play a direct role in making it happen, by communicating information about computer use directly to developers. Issues of privacy and confidentiality will have to be worked out; in some environments it will not be possible. But the potential advantages are so great, for both computer developers and computer users, that efforts to find a mutually satisfactory arrangement will be amply rewarded.

ACKNOWLEDGMENT

Over the five years I have been collecting data and organizing these observations, Susan Ehrlich Rudman and Steve Poltrock have been invaluable colleagues. Michael Good, John Gould, Gary Perlman, and John Richards helped with specific issues. Wang Laboratories, MCC, and Aarhus University provided the key resource, which is time.

REFERENCES

Aucella, A.F. (Mod.) (1987). Voice: technology searching for communication needs. In *Proc. CHI+GI'87 Human Factors in Computing Systems* (Toronto, April 5-9).

Bjerknes, G., Ehn, P. and Kyng, M. (Eds.) (1987). *Computers and democracy: A Scandinavian challenge.* Brookfield, VT: Gower Press.

Blomberg, J. (1986). The variable impact of computer technologies on the organization of work activities. In I. Greif (Ed.) *Computer-supported cooperative work: A book of readings.* San Mateo: Morgan Kaufmann, 1988.

Boehm, B. (1988). A spiral model of software development and enhancement. *IEEE Computer, 21,* 5, 61-72.

Byte Magazine (1989). Speech more important interface than graphics, Media Lab's Negroponte tells SIGGRAPH. *Byte,* November, 1989, p. 26.

Curtis, B., Krasner, H., and Iscoe, N. (1988) A field study of the software design process for large systems. *Communications of the ACM, 31,* 11, 1268-1287.

De Marco, T. (1978). *Structured analysis and system specification.* NY: Yourdon Press, Inc.

Ehn, P. (1988). *Work oriented design of computer artifacts.* Stockholm: Arbetslivcentrum.

Fairley, R.E. (1985). *Software engineering concepts.* NY: McGraw-Hill.

Friedman, A.L. (1989). *Computer systems development: History, organization and implementation.* Chichester, UK: Wiley.

Gentner, D.R. and Grudin, J. (1990). Why good engineers (sometimes) create bad interfaces. *Proc. CHI'90 Human Factors in Computing Systems*, (Seattle, April 1-4).

Good, M. (1985). The iterative design of a new text editor. In *Proc. of the Human Factors Society 29th Annual Meeting,* 571-574.

Gould, J.D. (1988). How to design usable systems. In M. Helander (Ed.) *Handbook of Human-Computer Interaction.* Amsterdam: North-Holland.

Gould, J.D., Boies, S.J., Levy, S., Richards, J.T., and Schoonard, J. (1987). The 1984 Olympic Message System: A test of behavioral principles of system design. *Communications of the ACM, 30,* 9, 758-769.

Gould, J.D. and Lewis, C.H. (1983). Designing for usability -- key principles and what designers think. In *Proc. CHI'83 Human Factors in Computing Systems,* 50-53.

Gould, J.D. and Lewis, C. (1985). Designing for usability: Key principles and what designers think. *Communications of the ACM, 28,* 3, 300-311.

Grudin, J. (1986). Designing in the dark: Logics that compete with the user. In *Proc. CHI'86 Human Factors in Computing Systems,* (Boston, April 13-17).

Grudin, J. (1988). Why CSCW applications fail: Problems in the design and evaluation of organizational interfaces. In *Proc. CSCW'88 Conference on Computer-Supported Cooperative Work* (Portland, September 26-28). Revision appeared as Why groupware applications fail: Problems in design and evaluation, *Office: Technology and People, 4,* 3, 1989, 245-264.

Grudin, J. (1989). The case against user interface consistency. *Communications of the ACM, 32,* 10, 1164-1173.

Grudin, J. (1990a). Groupware and cooperative work: Problems and prospects. In B. Laurel (Ed.) *The art of human-computer interface design.* Reading, MA: Addison-Wesley.

Grudin, J. (1990b). interface. In *Proc. CSCW'90 Conference on Computer-Supported Cooperative Work* (Los Angeles, October 7-10).

Grudin, J., (1990c). Seven plus one challenges for groupware developers. Manuscript submitted for publication.

Grudin, J., (1990d). The development of interactive systems: Bridging the gaps between developers and users. Manuscript submitted for publication.

Grudin, J., Ehrlich, S.F., and Shriner, R. (1987). Positioning human factors in the user interface development chain. In *Proc. CHI+GI'87 Human Factors in Computing Systems* (Toronto, April 5-9).

Grudin, J. and Poltrock, S. (1989). User interface design in large corporations: Communication and coordination across disciplines. In *Proc. CHI'89 Human Factors in Computing Systems,* (Austin, April 30-May 4).

Grønbæk, K. (1990). Supporting active user involvement in prototyping. *Scandinavian Journal of Information Systems, 2,* in press.

Grønbæk, K., Grudin, J., Bødker, S., and Bannon, L. (1990). Improving conditions for cooperative system design - shifting from a product to a process focus. In Schuler, D. and Namioka, A. (Eds.), *Participatory design.* In preparation.

International Herald Tribune (1990). Philips, facing losses, to trim 10,000 jobs. July 3.

Jackson, M. (1983). *System development.* Englewood Cliffs, NJ: Prentice-Hall.

Johnson, T. (1985). *Natural language computing: The commercial applications.* London: Ovum Ltd.

Kapor, M. (1987). Interview, *INC. Magazine,* January, 1987.

Landauer, T.K. (1988). Research methods in human-computer interaction. In M. Helander (Ed.), *Handbook of human-computer interaction.* Amsterdam: North Holland.

Leventhal, L.M. (1988). Experience of programming beauty: some patterns of programming experience. *Int. J. Man-Machine Studies, 28,* 525-550.

Lewis, C. and Polson, P. (1990). Theory-based design for easily learned interfaces. Manuscript.

Licklider, J.C.R. and Clark, W.E. (1962). On-line man-computer communication. *AFIPS Conference Proc. 21,* 113-128.

Mace, S. (1990). Defending the Dbase turf. *InfoWorld, 12,* 2 (January 8), 43-46.

McKendree, J. and Mateer, J. W., 1989. Film techniques applied to the design and use of intelligent systems. MCC Technical Report ACT-HI-261-89.

Mintzberg, H., 1984. A typology of organizational structure. In D. Miller and P.H. Friesen (Eds.), *Organizations: A quantum view.* Englewood Cliffs, N.J.: Prentice-Hall.

Mountford, J. (Moderator), 1989. Drama and personality in user interface design. In *Proc. CHI'89 Human Factors in Computing Systems.*

Negroponte, N., 1990a. The noticeable difference. In B. Laurel (Ed.) *The art of human-computer interface design.* Reading, MA: Addison-Wesley.

Negroponte, N., 1990b. Hospital corners. In B. Laurel (Ed.) *The art of human-computer interface design.* Reading, MA: Addison-Wesley.

Nelson, T.H., 1990. The right way to think about software design. In B. Laurel (Ed.) *The art of human-computer interface design.* Reading, MA: Addison-Wesley.

Norman, D.A. and Grudin, J. (1990) Conflicting guidelines reflect design tradeoffs. Manuscript.

Parker, R. (1990). Bill Lyons' task: Incremental moves to consistency. *InfoWorld, 12,* 2 (January 8), 44.

Perlman, G. (1989). Design/evaluation methods for hypertext technology development. In *Proc. Hypertext'89,* (Pittsburgh, Nov. 5-8), 61-81.

Polson, P. (1988). The consequences of consistent and inconsistent user interfaces. In R. Guindon (Ed.), *Cognitive science and its applications for human-computer interaction.* Hillsdale, NJ: Lawrence Erlbaum.

Poltrock, S.E. (1989a). Participant-observer studies of user interface design and development. MCC Technical Report ACT-HI-125-89.

Poltrock, S.E. (1989b). Innovation in user interface development: Obstacles and opportunities. In *Proc. CHI'89 Human Factors in Computing Systems,* (Austin, April 30-May 4).

Poltrock, S.E. (1989c). Participant-observer studies of user interface design and development: Communication and coordination. MCC Tech. Report ACT-HI-162-89.

Rubinstein, R. and Hersh, H. (1984). *The human factor.* Bedford, MA: Digital Press.

Shneiderman, B. (1980). *Software psychology: Human factors in computer and information systems.* Cambridge, MA: Winthrop.

Shneiderman, B. (1987). *Designing the user interface: Strategies for effective human-computer interaction.* Reading, MA: Addison-Wesley.

Smith, S.L. and Mosier, J.N. (1986). Guidelines for designing user interface software. *Report 7 MTR-10090, Esd-Tr-86-278*. Bedford, MA: MITRE Corporation.

Straub, D.W. and Wetherbe, J.C. (1989). Information technologies for the 1990s: An organizational impact perspective. *Communications of the ACM, 32,* 11, 1328-1339.

Whiteside, J., Bennett, J., and Holtzblatt, K. (1988). Usability engineering: our experience and evolution. In M. Helander (Ed.), *Handbook of human-computer interaction*. Amsterdam: Elsevier Science Publishers B.V. (North-Holland).

Williams, M. (1990). Panel statement, summarized in: The loyal opposition, *Proc. CHI'90 Human Factors in Computing Systems* (Seattle, April 1-5), p. 54.

Part of this paper will appear in *Proc. INTERACT'90 Conference on Human-Computer Interaction*. Versions of Part 1 that focus on Participatory Design and on Computer Supported Cooperative Work will be published in Schuler, D. and Namioka, A. (Eds.), *Participatory design*, Hillsdale, NJ: Lawrence Erlbaum Associates, and in *International Journal of Man-Machine Studies,* respectively.