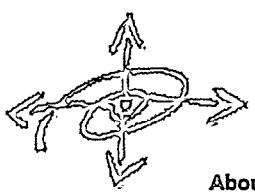
Brad A. Myers

Challenges of HCI Design and Implementation

Getting the user interface right is becoming critical to the success of products, and everyone knows that designing and implementing human-computer interfaces is difficult and time-consuming. But *why* is this true? Should we expect that a new method is around the corner that will make the design significantly easier? Will the next generation of user interface toolkits make the implementation trivial? No. This article discusses reasons why a focus on the user interface is important, and why user interface design and implementation are *inherently* difficult tasks and will remain so for the foreseeable future.



About the Author

BRAD MYERS *is a Senior Research Computer Scientist at Carnegie Mellon University, where he is the principal investigator for the Garnet User Interface Development Environment and the Demonstrational Interfaces Project. email: bam@cs.cmu.edu*

Why User Interfaces Are Important

A significant growth area for computers is their use in consumer electronics. This is why computer manufacturers like Apple are getting into the "personal digital assistant" market. The Friend21 project in Japan is a six-year project started in 1988 with the goal of promoting research and development into next-generation user interfaces, primarily intelligent agents and adaptive interfaces. It is funded at about $120 million, and is a consortium of 14 major Japanese companies organized by the Ministry of International Trade and Industry. Friend21 stands for Future Personalized Information Environment Development [26]. The project believes that in the 21st century everyone will be using computers for their everyday activities [26]. For the users of these devices, ease-of-use has become a prime factor in decisions about which ones to buy.

There is substantial empirical evidence that attention to usability dramatically decreases costs and increases productivity

Time is valuable, people do not want to read manuals, and they want to spend their time accomplishing their goals, not learning how to operate a computer-based system.

Usability has also become critical for commercial desktop software. User's demands on software have changed; they expect to be able to sit down and use software with little or no frustration. Readers of PC World magazine indicated in a survey that usability was as important a review parameter as the more traditional issues of speed and features. Thus, usability is a do-or-die decision for developers, and is being cited with increasing frequency and explicitness in product advertisements.

Although American industry has invested heavily in information technology, the expected productivity improvements have not been realized [2]. Usability at the individual, group and firm level has been cited as a culprit in this productivity paradox. For instance, the ever-changing computer environments caused by new product introductions and upgrades make continual learning demands on workers [2].

There is substantial empirical evidence that attention to usability dramatically decreases costs and increases productivity. A model of human performance, and a corroborating empirical study, predicted that a new workstation for telephone operators would *decrease* productivity despite improved hardware and software. The resulting decision not to buy the new workstation is credited with saving NYNEX an estimated $2 million a year [12].

A different study reported savings from usability engineering of $41,700 in a small application used by 23,000 marketing personnel, and $6,800,000 for a large business application used by 240,000 employees [17]. This was attributed to decreased task time, fewer errors, greatly reduced user disruption, reduced burden on support staff, elimination of training, and avoiding changes in software after release. One analysis estimates the mean benefit for finding each usability problem at $19,300 [19]. A mathematical model based on 11 studies suggests that using software which has undergone thorough usability engineering will save a small project $39,000, a medium project $613,000 and a large project $8,200,000 [24]. By estimating all the costs associated with usability engineering, another study found that the benefits can be up to 5000 times the cost [25].

Other studies have shown that it is important to have HCI specialists involved in design. A formal experiment reported that professional HCI designers created interfaces that had fewer errors and supported faster user execution than interfaces designed by programmers [3]. One reason is that training and experience in HCI design has a clear impact on the designer's mental model of interfaces and of the user interface design task [11]. This implies that HCI design is not simply a matter of luck or common sense, and that experience using a computer is not sufficient for designing a good user interface, but that specific training in HCI is required.

In addition, poor user interfaces have contributed to disasters including loss of life. For example, the complicated user interface of the Aegis tracking system was a contributing cause to the erroneous shooting down of an Iranian passenger plane, and the US Stark ship's inability to cope with Iraqi Exocet missiles was partly attributed to the human-computer interface [22]. Sometimes the *implementation* of the user interface can be at fault. A number of people died from radiation overdoses partially as a result of faulty cursor handling code in the Therac-25 [33].

The importance of a focus on human-computer interaction has been recognized by industry, academia, government and the trade press. The Committee to Assess the Scope and

Direction of Computer Science and Technology of the National Research Council in their report "Computing the Future" lists user interfaces as one of the six "core subfields" of CS, and notes that it is "very important" or "central" to a number of important application areas such as global change research, computational biology, commercial computing, and electronic libraries [15]. Two surveys of Information Services practitioners and managers listed Human Interface technologies as the most critical area for organizational impact [13]. New regulations, such as Directive 90/270 from the Council of European Communities, are being passed that require interfaces to be "easy to use and adaptable to the operator" [5]. ACM has started two new publications about HCI: *Transactions on Computer- Human Interaction* and this magazine, *interactions*. ARPA and NSF in the United States, ESPRIT in Europe and MITI in Japan have all initiated significant HCI initiatives.

User Interfaces Are Hard to Design

Although the benefits of usability engineering are clear, no one believes that this solves the problem of making interfaces easy to use. However, there is surprisingly little attention to *why* user interfaces are difficult to design.

The Difficulty in Knowing Tasks and Users

The first command to user interface designers is "know thy user." This has been formalized to some extent by the HCI sub-field of "task analysis." Unfortunately, this is extremely difficult in practice.

Surveys of software in general show that the deep application-specific knowledge which is required to successfully build large, complex systems is held by only a few developers, and is hard to acquire [10].

Furthermore, Don Norman reports:

> *My experience is that the ... initial specifications ... are usually wrong, ambiguous or incomplete. In part, this is because they are developed by people who do not understand the real problems faced by the eventual users.... Worse, the users may not know what they want, so having them on the design team*

No Silver Bullet

Like software in general, there is no "silver bullet" [7] to make user interface design and implementation easier. It seems that user interfaces are often more difficult to engineer than other parts of a system. For example, in addition to the difficulties associated with designing any complex software system, user interfaces add the following problems:

- **designers have difficulty thinking like users**
- **tasks and domains are complex**
- **various aspects of the design must be balanced (standards, graphic design, technical writing, internationalization, performance, multiple levels of detail, social factors, legal issues, and implementation time)**
- **existing theories and guidelines are not sufficient**
- **user Interface design is a creative process**
- **iterative design is difficult**

User interfaces are especially hard to implement because:

- **they are hard to design, requiring iterative implementation**
- **they are reactive and must be programmed from the "inside-out"**
- **they generally require multiprocessing**
- **there are real-time requirements for handling input events**
- **the software must be especially robust while supporting aborting and undoing of most actions**
- **it is difficult to test user interface software**
- **today's languages do not provide support for user interfaces**
- **the tools to help with user interfaces can be extremely complex**
- **programmers report an added difficulty in modularizing user interface software**

is not a solution. Actually, developing correct specifications may not be solvable, because ... a true understanding of a tool can only come through usage, in part because new tools change the system, thereby changing both needs and requirements... All the formalization in the world will not help us solve this problem. [electronic mail message]

The user interface portion of the code requires an even deeper understanding of the users than the design of the functionality since the interface must match the skills, expectations and needs of the intended users. Users are extremely diverse, so interfaces good for some may be bad for others. The "individual differences" sub-field of HCI is devoted to studying this problem. Furthermore, designers can never anticipate all the different uses to which the system will be applied.

There is ample evidence that programmers have a difficult time thinking like end-users [11]. One inherent difficulty is that programmers and designers cannot remember what they used to not know. Experiments have shown that people are unable to return in memory to their novice state [8]. Hence they cannot anticipate the reactions of novices and overestimate what novices actually know. Furthermore, one of the biggest failings of bad user interfaces is that they require users to think in terms of system objects and concepts rather than in the objects and concepts of the application domain. HCI specialists seem to be better at thinking like end users, which is one reason their interface designs are easier to use. But finding HCI specialists who are also domain experts is often difficult.

The Inherent Complexity of Tasks and Applications

An ordinary telephone is pretty easy to use, but modern business phones that can hold, transfer, record, and playback calls can be quite challenging due to the increased complexity. Similarly, Microsoft Word for the Macintosh has about 300 commands and CAD programs like AutoCAD have over 1000. It is clearly impossible for applications with that many functions to have an interface that is as easy to learn and use as one that has only a few functions.

This increased complexity comes from many sources. Partly, it results from the complex requirements in the domain itself. For example, CAD programs must provide techniques for carefully aligning objects, which is not necessary in simple drawing packages. Another reason is that each new version of a product needs to have new features so people will be motivated to upgrade. Additional complexity arises from providing a single, generic application that must work for a variety of users and domains. Thus, Microsoft Word has dozens of ways to move the cursor, so that individuals' preferences can be accommodated. Similarly, CAD programs might provide a dozen different ways to draw a circle so that users can choose the appropriate method for their tasks.

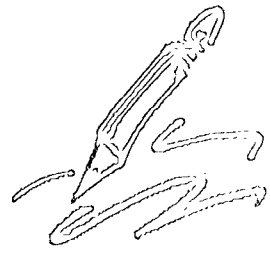
One way to try to overcome complexity is to use metaphors that exploit the user's prior knowledge by making interface objects seem like objects that the user is familiar with. However:

instead of reducing the absolute complexity of an interface, this approach seeks to increase the familiarity of the concepts.... [However] the inevitable mismatches of the metaphor and its target are a source of new complexities for users. [9]

The Variety of Different Aspects and Requirements

All design involves tradeoffs, but it seems that user interface design involves a much larger number of concerns, and they are the purview of widely different disciplines. User interface design includes considerations about:

Standards: An interface will usually need to adhere to standard user interface guidelines, such as the Macintosh, Windows, or Motif user interface styles. However, these style guides are usually hard to interpret and apply. Furthermore, the standards will only cover a small part of the user interface design, and will not insure that even this part has high usability. Other "standards" with which a design might need to be compatible include previous versions of the product, and related products from competitors.



Graphic design: An important part of the user interface design is the graphical presentation, including the layout, colors, icon design,

and text fonts. This is typically the province of professional graphic designers.

Documentation, messages, and help text: One study showed that rewriting the help messages, prompts, and documentation to increase their quality had significantly more impact on the usability of a system than varying the interface style [6]. Thus it is important to have good technical writers participating in the design.

Internationalization: Many products today will be used by people who speak different languages. Internationalizing an interface is much more difficult than simply translating the text strings, and may include different number, date, and time formats, new input methods, redesigned layouts, different color schemes, and new icons [28].

Performance: Users will not tolerate interfaces that perform too slowly. For example, it was reported that users did not like early versions of the Xerox Star office workstation because there were delays in the response time, even though the users' overall productivity was much higher. Performance concerns explain why moving windows on the Macintosh shows XORed outlines rather than having the entire window move as on the NeXT. The designer must always balance what is desirable with what will keep up with the mouse.

High-level and low-level details: It is not sufficient to get the overall model correct; each low-level detail must also be perfected. If users do not like the placement of the "control" key on the keyboard, or cannot find a menu item, they will not like the interface. Similarly, even if each low-level detail is perfect, if the overall system model does not make sense, the interface may be unusable.

External factors: Many systems fail for political, organizational, and social reasons entirely independent of the design of the interface. If users perceive that the software will threaten their jobs or status, they will not like it no matter what the user interface. Designers should take into account the social context in which their system will be used, and try to involve users in the system's design so they will feel less threatened.

Legal issues: One way to get a good design is to copy a design that has proven to be workable and popular. Unfortunately, there are many situations where this is illegal today. Lotus sued Paperback Software for copying its menu structure, and Apple has sued a number of companies for copying its user interface. Designers must be aware of which interface elements can be used and which cannot.

Time to program and test: There is always a trade-off between the time to test and perfect a user interface, and the time to ship the product. The more times an interface is iteratively refined, the better it is likely to be, but then it will be later to reach the marketplace.

Others: Interfaces that are aimed at special audiences have additional concerns. For example, software that helps multiple users collaborate (computer-supported cooperative work or CSCW) has interesting design constraints, such as what does Undo mean when multiple people are using the same software? Advanced input devices and techniques, such as pen-based gesture recognition, speech, or Data Gloves™, also raise many interesting issues.

The implication of these requirements is that all user interface design involves trade-offs, and it is impossible to optimize all criteria at once. Since one person would find it difficult to be competent, let alone expert, in this many areas, multiple people with quite different skills must be involved with different parts of the design. This increases the coordination and management overhead. It may be especially difficult since people from different backgrounds often have different terminology and approaches to problems.

Theories and Guidelines Are Not Sufficient

There are many methodologies, theories and guidelines for how to produce a good user interface (each ACM CHI conference proceedings is likely to have a few). Some of these guidelines are quite specific (e.g., "do not use more than three fonts"), while others are quite vague (e.g., "minimize the amount of input from the user"). Smith and Mosier have compiled 944 guidelines in a 478 page report [31]. Although there are a number of reports of successful systems created using various methodologies, evidence suggests that the skill of the designers was the primary contributor to the quality of the interface, rather than the method or theory. In fact, there are important counter-examples to even

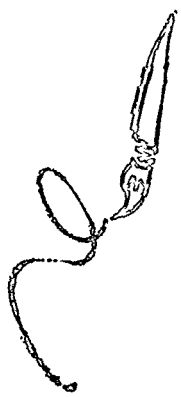
the most basic guidelines. For instance, most sources put consistency at the top of lists of guidelines, but Grudin discusses many cases where consistency is not appropriate. For example, menu systems might have the default selection be the more recent or most likely selection, but still might not use this rule for questions confirming dangerous operations [14]. In addition, some of the guidelines in Smith and Mosier are contradictory.

Whereas early papers in HCI were full of experimental laboratory studies of small issues in user interface design, such as the proper menu organization, you rarely see any of these now because the results have failed to generalize.

In fact, Tom Landauer says:

For the most part, useful theory [from cognitive psychology] is impossible, because the behavior of human- computer systems is chaotic or worse, highly complex, dependent on many unpredictable variables, or just too hard to understand. Where it is possible, the use of theory will be constrained and modest, because the theories will be imprecise, will cover only limited aspects of behavior, ... and will not necessarily generalize. [18]

Of course, other researchers disagree. For example, current research on modeling users with the GOMS model has successfully helped evaluate interfaces and predict human behavior in a large and ever- growing number of circumstances: text-editing, VLSI layout, graphical editing, spreadsheets, computer command abbreviations, high-functionality oscilloscopes, telephone operator workstations, video games, etc. [16]



Design is a Creative Process

As a result of the lack of theory and methodology, user interface design remains a creative process, rather than a mechanized process of following rules. In fact, many consider designing user interfaces to be more like creating works of art rather than the product of proper engineering. Thus, user interface design may be more like architectural design, or even photography, where there are significant technical skills and rules that must be learned, but fundamentally the design is artistic. And as with these other creative activities, some people will have more talent for them than others. Whereas courses can certainly teach people important lessons that may bring their user interface designs to a level of competence, it may be impossible to teach how to make great designs, just as photography courses cannot teach students how to be the next Ansel Adams.

The Difficulty of Iterative Design

Due to the difficulties just described, HCI professionals and HCI methodologies recommend iterative design, where the interface is prototyped and repeatedly redesigned and tested on actual end users. A recent survey reported that 87% of the development projects used iterative design in some form [21]. (Other surveys put the number at around 50%.) However, this process is also quite difficult.

One important problem is that the designer's intuition about how to fix an observed problem may be wrong, so the new version of the system may be worse than the previous version. Therefore, it is difficult to know when to stop iterating. Furthermore, "... [experimental] data supports the idea that changes made to improve one usability problem may introduce other usability problems" [3]. The same data also showed that while iterating on a poor design does improve it, iteration never gets it to be as good as an interface that was originally well-designed. Thus iterative design does not obviate the need for good designers.

Another important problem is getting "real" users with which to test. "Too often ... testers have to extrapolate from 'problem' users who bring a set of 'hidden agendas' with them to the test session" [4]. The actual users of a product may be different from the buyers, so it is important not to use the buyers as subjects. Participants in tests are usually self-selected, so they are likely to be more interested, motivated, and capable than the actual end users. Furthermore, when users know they are participating in a test, they often behave differently than they would in natural use of the system. Each iteration of the testing should involve different users, so a large number of people might be needed.

Finally, iterative testing can be quite long and expensive. Formal tests may take up to 6 weeks,

so getting answers back to the design team may be slow. A usability lab may cost between $70,000 and $250,000 in capital costs to set up, plus professional staff. When contracted out to a consulting firm, a single usability test may cost between $10,000 and $60,000, and when performed in house, $3000 to $5000 [1]. For CSCW systems which are used to link multiple people, user testing is especially difficult because a realistic task usually requires multiple people and significant lengths of time. Nielsen provides a survey of the costs for various techniques [24], and shows that the benefits outweigh the costs. Still the costs are considerable, and can take a long time, which conflicts with the desire to get products out quickly.

User Interfaces Are Hard to Implement

Many surveys have shown that the user interface portion of the software accounts for over half of the code and development time. For example, one survey reports that over a wide class of program types, machine types and tools used, the percent of the design time, the implementation time, the maintenance time, and the code size devoted to the user interface was about 50% [21]. In fact, there are a number of important reasons why user interface software will inherently be among the most difficult kinds of software to create. For example, if you list the general properties that will make any system difficult to implement, multiprocessing, robustness and real-time requirements will be at the top of the list, and these are all often present in user interface software.

Need for Iterative Design

The need to use iterative design means that the conventional software engineering "waterfall" approach to software design, where the user interface is fully specified, then implemented, and later tested, is inadequate. Instead, the specification, implementation, and testing must be intertwined [32]. This makes it very difficult to schedule and manage user interface development.

Reactive Programming

Once the implementation begins, there are a number of properties of user interface software that make it more complex than other kinds of software, especially for graphical, window-based interfaces. One big difference is that modern user interfaces must be written "inside-out." Rather than structuring the code so that the application is in control, as is usually taught in computer science classes, the application must instead be structured as many subroutines which are called by the user interface tool kit when the user does something. This is sometimes called "event-based programming." Each subroutine will have stringent time constraints so that it will complete before the user is ready to give the next command. Programmers must be trained to write programs in this way, and it appears to be more difficult for programmers to organize and modularize reactive programs [27].

Multiprocessing

A related issue is that in order to be reactive, user interface software is often organized into multiple processes. All window systems and graphical tool kits queue "event" records to deliver the keyboard and mouse inputs from the user to the user interface software. Users expect to be able to abort and undo actions (for example, by typing control-C or Command-dot). Also, if a window's graphics need to be redrawn by the application, the window system notifies the application by adding a special "redraw" event to the queue. Therefore, the user interface software must be structured so that it can accept input events at all times, even while executing commands. Consequently, any operations that may take a long time, such as printing, searching, global replace, re-paginating a document, or even repainting the screen, should be executed in a separate process. Alternatively, the long jobs could poll for input events in their inner loop, and then check to see how to handle the input, but this is essentially a way to simulate multiple processing. Furthermore, the window system itself often runs as a separate process. Another motivation for multiple processes is that the user may be involved in multiple ongoing dialogs with the application, for example, in different windows. These dialogs will each need to retain state about what the user has done, and will also interact with each other.

Therefore, programmers creating user interface software for these window systems and tool kits will usually encounter the well-known

problems with multiple processes, including synchronization, maintaining consistency among multiple threads, deadlocks, and race conditions.

The Need for Real-time Programming

Another set of difficulties stems from the need for real-time programming. Most graphical, direct manipulation interfaces will have objects that are animated or which move around with the mouse. In order for this to be attractive to users, the objects must be redisplayed between 30 and 60 times per second without uneven pauses. Therefore, the programmer must ensure that any necessary processing to calculate the feedback can be guaranteed to finish in about 16 milliseconds. This might involve using less realistic but faster approximations (such as XORed bounding boxes), and complicated incremental algorithms that compute the output based on a single input which has changed, rather than a simpler recalculation based on all inputs.

The next generation of user interfaces will include new technologies such as video, speech and other sounds, animations of simulations, and other "multimedia," all of which have quite stringent real-time constraints. The best way for programmers to control the temporal aspects of programs is still a difficult research question.

Need for Robustness

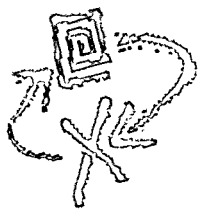
Naturally, all software has robustness requirements. However, the software that handles the users' inputs has especially stringent requirements because all inputs must be gracefully handled. Whereas a programmer might define the interface to an internal procedure to only work when passed a certain type of value, the user interface must always accept any possible input, and continue to operate. Furthermore, unlike internal routines that might abort to a debugger when an erroneous input is discovered, user interface software must respond with a helpful error message, and allow the user to start over or repair the error and continue. To make the task even more difficult, user interfaces should allow the user to abort and undo operations. Therefore, the programmer should implement most actions in a way that will allow them to be aborted while executing and reversed after completion. For example, all code that calls functions that ask the user for input must be prepared to accept a special return value that means the user aborted and did not provide the input. Special data structures and coding styles are often required to support this.

Low Testability

A related problem is the difficulty of testing user interface software for correctness. It is extremely difficult to determine whether the user interface has been tested completely. While all complex software is difficult to test, one reason that user interface software is more difficult is that automated testing tools are rarely useful for direct manipulation systems, since they have difficulty providing input and testing the output. For "regression testing" (to see if a new version of the software breaks things that used to work in the previous version), tools for conventional software will supply inputs and test the outputs against the values produced by the previous version. However, in a direct manipulation system, if buttons have moved or new items have been added to menus, a transcript of the input events from the previous version may not invoke the desired operations. Furthermore, the outputs of most operations are changes to the screen, which can be impossible for an automatic program to compare to a saved picture since at least something in each screen is likely to have changed between versions.

No Language Support

Another reason that programming user interface software is difficult is that the programming languages used today do not contain the appropriate features. For example, no popular computer programming language contains primitives for graphical input and output. Many languages, however, have input-output primitives that will read and write strings; for example, C provides scanf and printf. Unfortunately, using these procedures produces very bad user interfaces, since the user is required to answer questions in a highly-modal style, and there are no facilities for undo or help. Therefore, the built-in input/output facilities of the languages must be ignored and large external libraries must be used instead.



Iterative design is vital for good user interfaces, although it cannot replace having good designers.

As discussed before, user interface software is reactive and often requires multiprocessing. Features to support these are missing from programming languages. Research into user interface software has identified other language features that can make the creation of user interface software easier. For example, most people agree that user interface software should be "object-oriented" but languages do not seem to provide an appropriate object system: Apple had to invent Object Pascal to implement the first version of their MacApp framework, and the implementors of Motif and Open Look for Unix could not find an acceptable object system so they hacked together an object system into C called xt. One reason C++ is gaining in popularity is the recognized need for an object-oriented style to support user interface programming, but C++ has no graphics primitives or support for multiprocessing or reactive programming. I have just completed a book that discusses at length languages for programming user interfaces [20].

Complexity of the Tools

Since the programming languages are not sufficient, a large number of tools have been developed to address the user interface portion of the software. Unfortunately, these tools are notoriously difficult to use. Manuals for the tools often run to many volumes and contain hundreds of procedures. For example, the Macintosh ToolBox manuals now fill six books. Some tools even require the programmer to learn an entirely new special-purpose programming language to create the user interface (e.g., the UIL language for defining screen layouts for Motif). Clearly, enormous training is involved in learning to program user interfaces using these tools. In spite of the size and complexities of the tools, they may still not provide sufficient flexibility to achieve the desired effect. For example, in the Macintosh and Motif tool kits, it is easy to have a keyboard accelerator that will perform the same operation as a menu item, but very difficult to have a keyboard command do the same thing as an on-screen button.

It may also be difficult to use the underlying graphics packages, which allow the rectangles, circles and text to be drawn. Since the human eye is quite sensitive to small differences, the graphic displays must essentially be perfect: a single pixel error in alignment will be visible. Most existing graphics packages provide no help with making the displays attractive.

Difficulty of Modularization

One of the most important ways to make software easier to create and maintain is to appropriately modularize the different parts. The standard admonition in textbooks is that the user interface portion should be separated from the rest of the software, in part so that the user interface can be easily changed (for iterative design). Unfortunately, programmers find in practice that it is difficult or impossible to separate the user interface and application parts [27], and changes to the user interface usually require reprogramming parts of the application also. Furthermore, modern user interface tool kits make this problem harder because of the widespread use of "call-back" procedures. Usually, each widget (such as menus, scroll bars, buttons, and string input fields) on the screen requires the programmer to supply at least one application procedure to be called when the user operates it. Each type of widget will have its own calling sequence for its call-back procedures. Since an interface may be composed of thousands of widgets, there are thousands of these procedures, which tightly couples the application with the user interface and creates a maintenance nightmare [21].

Implications

Some of the implications of these results are clear. Developers designing user interfaces should involve trained user interface specialists, since they have proven to significantly improve the interfaces and be cost effective. Graphic

designers and technical writers should also be involved. While it is important to follow any guidelines that are relevant, this is not sufficient to ensure high quality. Iterative design is vital to the creation of good user interfaces, although it cannot replace having good designers. In general, usability engineering methods should be followed, especially since there are some simple "discount" methods that are often sufficient [23].

When implementing user interfaces, programming at the tool kit level is quite difficult, but there are a growing number of higher-level tools which can help significantly, and you should take advantage of these. For example, there are "interface builders" (which interactively lay out widgets) for every platform, and tools like Apple's HyperCard and Microsoft's Visual Basic can make prototyping and creation of some interfaces much easier. In fact, user interface tool kits are one of the few examples of large, extensively reusable, platform independent, portable libraries. Interactive UI tools are one of the only kinds of tools to have demonstrated the long-sought factor-of-ten improvement in programmer productivity. Tools that are coming out of research labs are covering increasingly more of the user interface task, are more effective at helping the designer, and are creating better user interfaces. These will probably evolve into commercial products fairly rapidly.

The implications for educators are also clear. The ACM, IEEE and others have called for Human-Computer Interaction to be a more central part of every computer programmer's education, since estimates are that more than half of programmers will be dealing with user interfaces. At the very least, all programmers need to understand that HCI design is a valid subfield where special training is required to create high-quality user interfaces. Also, there needs to be more programs where HCI specialists can be trained [30].

The agenda for researchers has been exhaustively surveyed elsewhere [29]. There are opportunities in new interaction styles and input devices (especially for manipulating 3-D objects), user interface software tools (especially for model-based and demonstrational construction of interfaces), better processes and methodologies for developing and evaluating user interfaces (especially those based on testable models), and interfaces for special applications such as those involving multiple users (computer-supported cooperative work), extremely large collections of data (searching, browsing, visualizing), and new domains.

While the design and implementation of all complex software is difficult, user interfaces seem to add significant extra challenges. We can expect research into user interface design and implementation to continue to provide better theories, methodologies and tools, but the problems discussed in this article are not likely to be solved, and the user interface portion will continue to be difficult to design and implement. Furthermore, as new styles of human-computer interaction evolve, such as speech and gesture recognition, intelligent agents, and 3-D visualization, the amount of effort directed to the design and implementation of the user interface can only increase. Fortunately, the research and development community is up to this challenge, and we can expect exciting innovations in user interface designs and software in the future. ■

Acknowledgment

Thanks to Robert Burns, Preston Ginsburg, Dario Giuse, Michael Gleicher, Bill Hefley, Bonnie John, Sara Kiesler, James Landay, Jon Meads, Bernita Myers, Jakob Nielsen, Frank Ritter, Bruce Sherwood, David Steier, Brad Vander Zanden, and Alan Wexelblat who provided helpful comments on earlier drafts of this paper.

This research was sponsored by the Avionics Lab, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U. S. Air Force, Wright-Patterson AFB, OH 45433-6543 under Contract F33615-90-C-1465, Arpa Order No. 7597.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

References

[1] Abelow, D. "Wake up! You've entered the eransition zone". Comput. Lang.10, 3 (Mar.. 1993), 41-47.

[2] Attewell, P. Information technology and the productivity paradox. In Organizational Linkages and Productivity. National Academy of Sciences, Washington D.C., 1993.

[3] Bailey, G. Iterative methodology and designer training in human-computer interface design. In Human Factors in Computing Systems: Proceedings of INTERCHI '93 (Amsterdam, The Netherlands, Apr.) ACM, New York, 1993, pp.198-205.

[4] Ballman, D. User involvement in the design process: Why, when and how. In Human Factors in Computing Systems: Proceedings of INTERCHI'93 (Amsterdam, The Netherlands, Apr.) ACM, New York, 1993, pp. 251-254.

[5] Billingsley, P. 1990 "EC directive may become driving force". SIGCHI Bull. 25, 1 (Jan. 1993), 14-18.

[6] Borenstein, N. The design and evaluation of on-line help systems. Ph.D. Thesis, Tech. Rep. CMU CS-85-151, Computer Science Dep., Carnegie Mellon Univ., Pittsburgh, Pa., 1985.

[7] Brooks, F.P.,Jr. "No silver bullet; Essence and accidents of software engineering". IEEE Comput. 20,

4 (Apr.1987), 10-19.

[8] Camerer, C., Loewenstein, G., and Weber, M. "The curse of knowledge in economic settings: An experimental analysis." J.Pol.Econ. 97(1989), 1232-1254.

[9] Carroll, J.M., Mack, R.L., and Kellogg, W.A. Interface metaphors and user interface design. In Handbook of Human-Computer Interaction. Elsevier Science Publishers B.V. (North Holland), 1988, pp. 67-85.

[10] Curtis, B., Krasner, H., and Iscoe, N. "A field study of the software design process for large systems." Commun. ACM 31, 11 (Nov. 1988), 1268-1287.

[11] Gillan, D.J., and Breedin, S. D. Designers' models of the human-computer interface. In Human Factors in Computing Systems: Proceedings of SIGCHI'90 (Seattle, Wa, Apr.) ACM, New York, 1990, pp. 391-398.

[12] Gray, W.D., John, B.E., and Atwoodm M.E., The precis of Project Ernestine, or an overview of a validation of GOMS. In Human Factors in Computing Systems: Proceedings of SIGCHI'92, Monterey, Ca., May) ACM, New York, 1992, pp. 307-312.

[13] Grover, V. and Goslar, M. "Information technologies for the 1990s: The executives' view". Commun. ACM 36, 3 (Mar. 1993), 17-19,102-103.

[14] Grudin. J. "The case against user interface consistency." Commun. ACM 32, 10 (Oct. 1989), 1164-1173.

[15] Hartmanis, J., et.al. "Computing the future." Commun. ACM 35, 11 (Nov. 1992), 30-40.

[16] John, B.E., Vera, A.H., and Newell, A., Towards real-time GOMS. In The Soar Papers: Research on Integrated Intelligence. MIT Press, Cambridge, Mass., 1993.

[17] Karat, C.-M. Cost-benefit analysis of usability engineering techniques. In Proceedings of the Human Factors Society 34th Annual Meeting. Vol. 2. Human Factors Society, 1990.

[18] Landauer, T.K. Let's get real: A position paper on the role of cognitive psychology in the design of humanly useful and usable systems. In Designing Interaction. Cambridge University Press, Cambridge, Mass., 1991, pp. 60-74.

[19] Mantei, M. M., and Teorey, T.J. "Cost/benefit analysis for incorporating human factors in the software lifecycle." Commun. ACM 31, 4 (Apr. 1988), 428-439.

[20] Myers, B.A., Ed. Languages for Developing User Interfaces. Jones and Bartlett, Boston, Mass., 1992.

[21] Myers, B.A. and Rosson, M. B. Survey on user interface programming. In Human Factors in Computing Systems: Proceedings of SIGCHI'92 (Monterey, Ca,, May)ACM, New York, 1992, pp. 195-202.

[22] Neumann, P.G. "Inside risks: Putting on your best interface." Commun. ACM 34, 3 (Mar. 1991).

[23] Nielsen, J. "Big paybacks from 'discount' usability engineering." IEEE Softw. 7, 3 (May 1990), 107-108.

[24] Nielsen, J. and Landauer, T.K. A mathematical model of the finding of usability problems. In Human Factors in Computing Systems: Proceedings of INTERCHI'93 (Amsterdam, The Netherlands, Apr.) ACM. New York, 1993, pp. 206-213.

[25] Nielsen, J. and Phillips, V.K.. Estimating the relative usability of two interfaces: Heuristic, formal and empirical methods compared. In Human Factors in Computing Systems: Proceedings of INTERCHI'93 (Amsterdam, The Netherlands, Apr.) ACM. New York, 1993, pp. 214-221.

[26] Nonogaki, H., and Ueda, H. FRIEND21 Project: A construction of 21st century human interface. In Human Factors in Computing Systems: Proceedings of SIGCHI'91 (New Orleans, La., Apr.). ACM, New York, 1991, pp. 407-414.

[27] Rosson, M.B., Maass, S., and Kellogg, W.A. Designing for designers: An analysis of design practices in the real world. In Human Factors in Computing Systems: CHI+GI'87 (Toronto, Ont., Canada, Apr.). ACM, New York, 1987, pp. 137-142.

[28] Russo, P. and Boor, S. How fluent is your interface? Designing for international users. In Human Factors in Computing Systems: Proceedings of INTERCHI'93 (Amsterdam, The Netherlands, Apr.) ACM. New York, 1993, pp. 342-347.

[29] Sibert, J., and Marchionini, G. "Human-computer interaction research agendas." Behav. Inf. Tech. 12, 2 (Mar.- Apr. 1993), 67-135.

[30] Hewett,T.T., Ed. ACM SIGCHI Curricula for Human-Computer Interaction. ACM Press, New York 1992.

[31] Smith, S.L., and Mosier, J.N. Guidelines for designing user interface software. Tech. Rept. ESD-TR-86-278, MITRE, Bedford, Mass., 1986.

[32] Swartout, W., and Balzer, R. The inevitable intertwining of specification and implementation. Commun. ACM 25, 7 (July 1982), 438-440.

[33] Levenson, N.G., and Turner, C.S. An investigation of the Therac-25 accidents. IEEE Comput. 26, 7 (July 1993), 18-41.