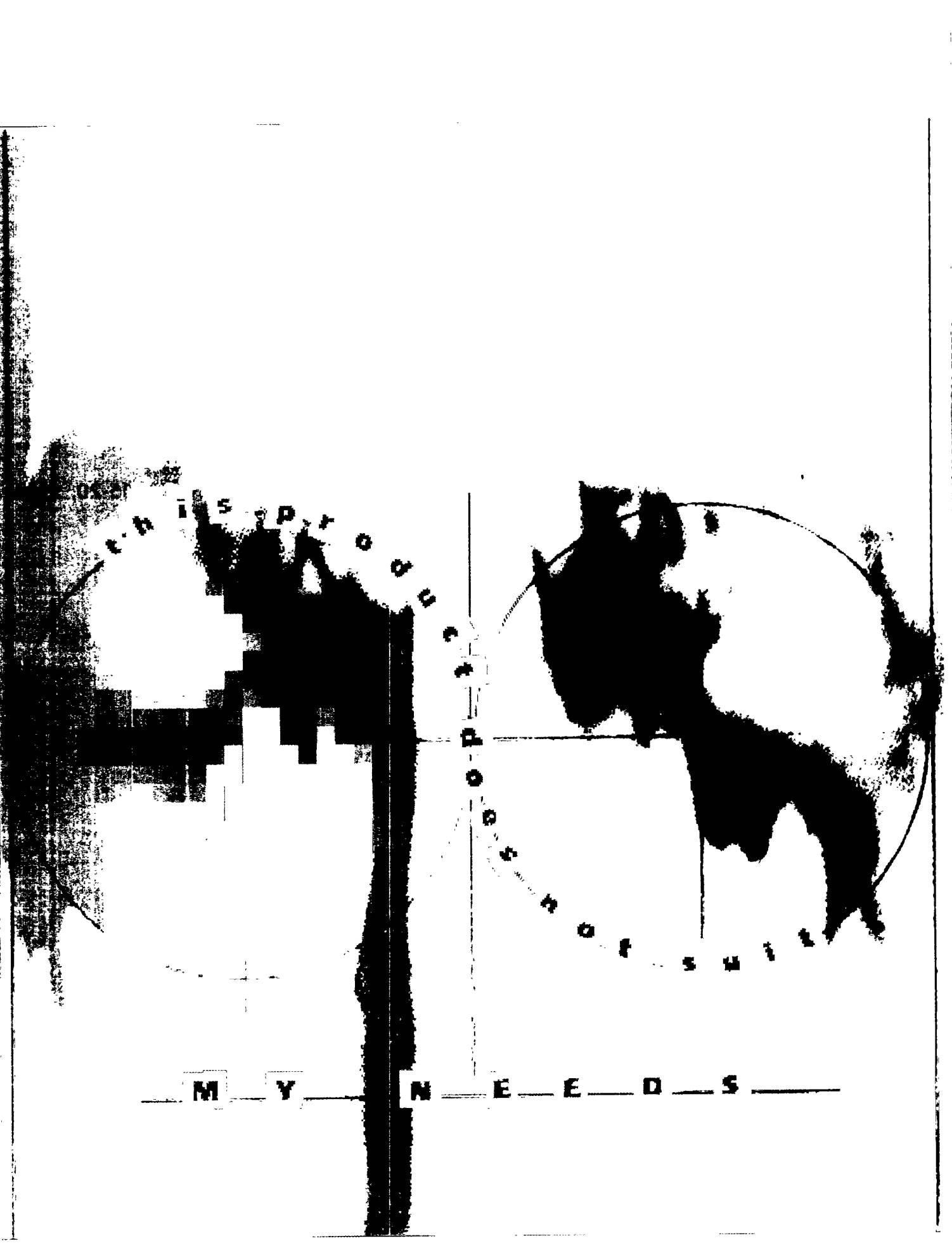
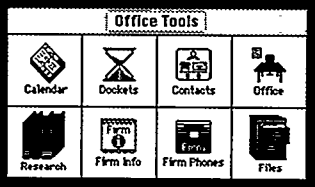

this product
does not suit
MY NEEDS

Michael Sellers

User-centered design is gaining acceptance with some software manufacturers, but it is still an immature, heuristically driven craft: We have a few principles of product analysis, design, and evaluation, but most of what we do is guided by personal intuition and experience — not to mention being constrained by chronically too-short schedules. As I try to improve my methods, I have found that developing highly usable interfaces for demanding users provides an acid test for user-centered design processes. Understanding demanding users and products made for them helps underscore what we know about interaction design, what we have yet to figure out, and what we need to provide in the future.



Designing for Demanding Users

About the Authors

MICHAEL SELLERS *is founder of New World Designs, a user interface design consulting firm, and has been designing user interfaces for over ten years. He has a BS in Cognitive Science, and is a very demanding user.*
E-mail: sellers@acm.org

Demanding Users

There is a simplistic model of the "user" in many software design circles: Anyone who is too dumb to write software merely uses it, and thus is a user. As one software support person said of conversations with customers, "It helps if I just imagine I'm talking to Homer Simpson." These hapless folks are too often seen as passive consumers, and many manufacturers give little consideration to their actual tasks, needs, and desires.

The software industry has operated in high gear for over a decade with this point of view. The impenetrable cult of high technology and the lack of alternative sources have long kept the software-consuming masses from revolting. After all, when one word processor or spreadsheet is as bizarre as another, what's the point in complaining about usability? The flip side of this is that customers who have invested large amounts of time in learning a product become rabidly loyal to it. It may still be awkward and confusing, but it's a confusion they already know. (Note that uninitiated end-users are not the only ones in this boat: Despite the advent of WYSIWYG editors and graphical symbolic debuggers, an easy way to pick a fight in a group of software engineers is to express a preference for one of the two old war horses vi and emacs. Both are horrendously unusable, yet the loyalty remains and the battles rage on.)

As the software industry expands, new products expose formerly non-computer literate groups to the joys of automation. However, some of these groups, notably doctors, lawyers, and many children, are unwilling to bow passively at the altar of software. Rather than spend their time conforming themselves to the software that programmers hand them, these users require that the products they use be engaging, informative, and effective from the beginning. Unlike traditional captive users, these demanding users are generally in control of how their software dollars are spent, and will not buy products that increase their cognitive load or that do not increase their capabilities immediately. It is these users that point the way we need to be moving — in designing human-computer interactions we need to design for the toughest audience, so that everyone can benefit.

Software

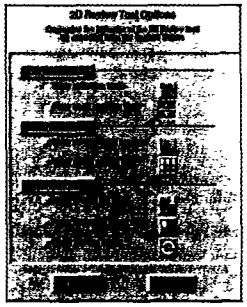
Products created with the demanding user in mind include visualization packages for the medical, scientific, and financial fields (e.g., ISG Technologies' Physician's Review Station); some information management products (e.g., Gavel & Gown's Amicus Attorney); and many video games and other entertainment products (notably the recent Living Books CD-ROM series). These projects demonstrate the importance of designing for the user, and that even when the user-centered design process is only partially implemented, it has a positive effect on the development and acceptance of real world products created for demanding users.

ISG's Physician's Review Station is a complex product that enables radiologists — intelligent, rushed, and not interested in "driving a computer" — to examine and manipulate images from medical scanners using any of several flexible, graphical tools. This Unix/Motif-based product fills the need for a low-cost way to improve and speed up radiological diagnoses.

As we performed our initial user and product analyses, we recognized the importance of enabling the doctors to come up to speed on the system quickly. We became aware of the trepidation many doctors felt towards new and ever more complex computer products. In particular, those over the age of about 45, who have both visual acuity that is beginning to diminish and a large voice in buying decisions, were reluctant to try new computer tools. In my investigations, I saw several situations where a doctor would not approach within about three feet of a computer system unless it looked especially attractive, and many who simply said, "This product does not fit my needs" — when in reality, it just appeared incomprehensible to them.

As a result, we came up with a "three-foot/five-minute" heuristic for initial user acceptance: The product had be attractive enough to get doctors inside the three-foot zone, and

As one software support person said of conversations with customers,



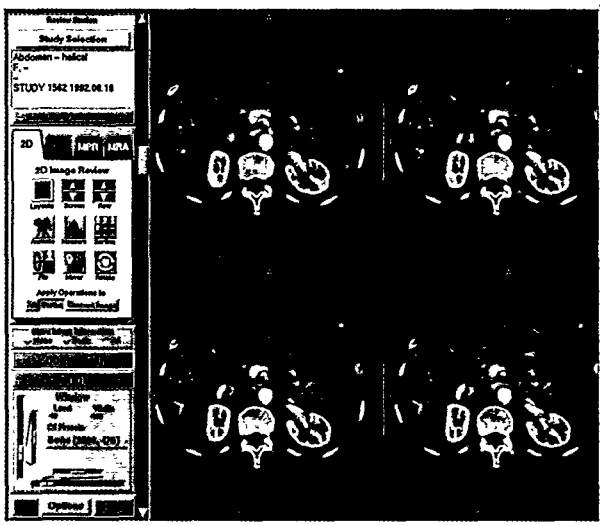
usable enough that any radiologist — even those who had not touched a typewriter keyboard in years — would be able to perform a diagnostically significant action within 5 minutes of starting to use the product. We did not expect full proficiency immediately, but this was nevertheless a major departure from earlier products that typically required 3-5 days of training.

This goal did not represent a formal usability test suite, and yet it guided many feature and interface design decisions and acted as a touchstone to the character of the entire project. As development proceeded, we came up with other usability goals (e.g., 1 hour to self-report of proficiency; 1 minute to select a patient and exam for the novice; 10-second patient selection for the familiar user) as a way of measuring our progress. Thus far we have met or exceeded most of our goals, including the most daring that specified such a short learning curve.

As part of our user analyses, we also examined the social and organizational context in which radiologists work and adapted the product to fit their needs. Specifically, most doctors will not use a product that puts them in a position of even appearing to be on unfamiliar ground, or which puts their social standing at risk. This point (and the steps we took to combat it) turned out to be a major underlying theme in our user interface design. We discovered that a serious consideration for many doctors was their ability to work in private, in particular away from administrators and medical technologists who occupy different social strata. This aspect of product use does not fall formally under the user interface, but it is definitely part of user-centered design. Had we not looked at the product from an organizational point of view, we would have missed the opportunity to skirt a major obstacle to user acceptance.

In designing this product's user interface, we were careful to understand and clearly represent the doctors' common and critical tasks by observing and interviewing potential users in their normal working environments. We thus eliminated many conceptual barriers the users had to getting started with this kind of product. To this end, the program initially presents a highly streamlined[1] set of graphical and direct manipulation tools organized into four major and clearly visible groups. This allows the user to begin using the system immediately, without spending time in formal training. The tools available at first are limited in number and scope, but correspond to those tasks most doctors perform most of the time. This limitation allows the doctor to get right to work, restricts the sources of confusion in the interface, and encourages the users to explore the product at their own pace.

Figure 1
ISG Technology's Physician's Review Station. Simple dialog boxes allow the user to tailor the interface to their needs.



In addition to the opening set of tools, we provide Options dialog boxes (Figure 1) that allow the user to easily alter their tool set as they wish. We used this idea of feature sets that are configurable without being intimidating throughout the product. It has met with strong user approval from both computer novices and expert users.

Figure 2
ISG Technology's Physician's Review Station. The interface uses muted colors and icons whenever possible. A certain elegance and medical professionalism is important to the product.

Not all of our ideas were successful, however. Early in the product development, we provided context-specific guidance text in one corner of the screen to help direct the user's actions. Unfortunately, we found during early testing that the text had only an interfering effect: The users did not read it for content, and yet its mere presence increased their cognitive load. We discovered that removing the text resulted in better user acceptance and did not

[1]Even the term "streamlined" was chosen with care, and with the users in mind. Doctors do not like to see themselves as doing anything simple, so words that evoke this — novice, basic, etc. — were avoided in the product.

"It helps if I just imagine I'm talking to Homer Simpson."

adversely affect performance. After this we concentrated on keeping the user interface streamlined enough that such guidance text would not be necessary.

By giving the product a polished, simple (almost austere) interface (Figure 2) we attended to the users' personal and social needs for clarity and medical professionalism. The high degree of interactivity in the product attended to their perceptual, cognitive, and task-oriented needs. Finally, by creating a lower-cost product that could sit on the doctor's desk, we avoided many organizational roadblocks to user acceptance. Even though our user-centered design process in this instance was more rough-and-ready than methodical, it was still a success. The Physician's Review Station is still undergoing FDA approval in the US, but has already set a new standard for usability in the medical imaging industry.

In another project I worked with a team at Gavel & Gown Software to develop the user interface for the lawyers' practice information management product Amicus Attorney. This Macintosh-based product is specifically suited to meet the demands of lawyers. Familiar items such as appointments, dockets, and client files make up the core of the product. They appear in a natural fashion that supports single-handed, point-and-click navigation whenever possible (the lawyer's other hand typically is holding a telephone receiver, document, or favorite pen). Even sound is used in a natural, complementary fashion to inform the user without annoying them: The sound of a wooden file drawer opening gives an aural cue that complements the visual changes on the screen when entering the client files "module," for example.

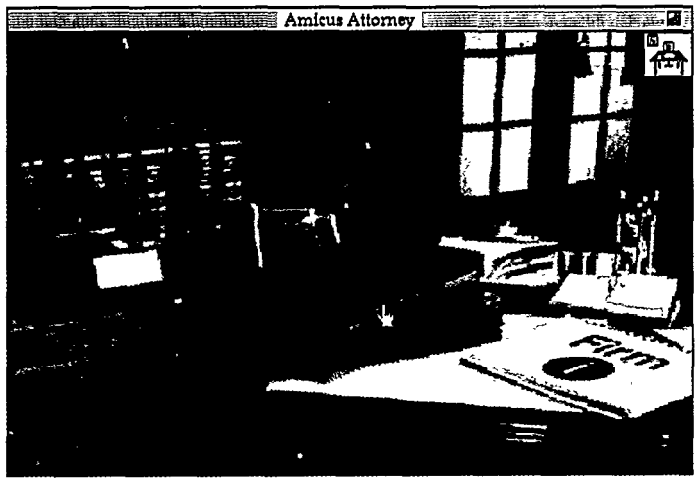


Figure 3
Gavel & Gown Software's Amicus Attorney. The user can choose their tools from either a standard iconic tool palette (see Figure 4) or from a realistic view of a lawyer's desk. In this picture the user can click on the books, the address file, the telephone, the clipboard, the hour glass, the calendar, the file folders, or either of the two piles of paper to access familiar functions. Shown here, the user is about to click on and open their client files. Clicking on the icon in the upper-right corner returns the user to the standard view.

We extended the concept of natural presentation in this product to allow the user to choose primary tools (files, calendar, dockets, telephone, contacts, etc.) from either a traditional iconic menu or from a digitized photograph of a lawyer's desk showing all these objects (Figure 3). This allows the user to work in whichever environment they find more appealing, thus supporting both the rank novice and the more traditional computer user. In addition, the colors and backgrounds used in the program evoke a legal setting without overpowering the product itself: golds, red leather, oak, and green marble are used judiciously throughout (Figures 4 and 5). While this might seem trivial, such attention to detail increases the users' emotional engagement with the product. In this case, it has increased user acceptance in an otherwise reluctant market — particularly among older lawyers. It provides users with a product that conforms to their world and expectations, and which can be on-screen at all times

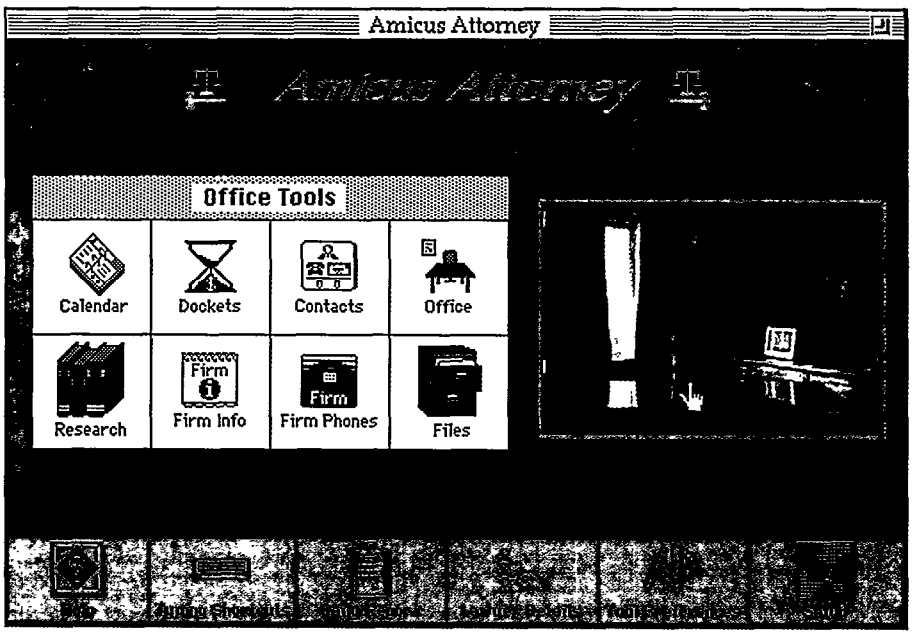


Figure 4:
Gavel & Gown Software's Amicus Attorney. This is the opening "splash" screen from an early Macintosh version based on Hypercard 2.2. Note the red leather in the background, and the general color scheme throughout. Clicking on the photo of the lawyer's office takes the user to the pictorial selector shown in Figure 3.

without diminishing their stature in front of a partner or client.

Like many startup products, the initial development of Amicus Attorney progressed informally, without a strong user-centered design method. Nevertheless, this product is an excellent example of an innovative, user-centered product developed for demanding users. In an earlier incarnation, this product was used internally at a major legal firm, and included a "Suggestions" utility. Over five hundred user suggestions were reviewed in the development of the current version of the product. In addition, expert knowledge, observation, focus groups, and fast-paced, highly iterative design characterized this product's development. It has been enthusiastically received by its intended users since before its first official release in April of this year.

A different sort of example of successfully designing for demanding users can be found in the CD-ROM Living Books series by Living Books, , formerly a division of Brøderbund. I have not worked on these, but I have enjoyed their creativity and have been impressed by how completely the graphics, sound, and attention to detail has entranced my children (my 3-year-old daughter learned to load the CD-ROM and move and click the mouse in just a few minutes while "reading" these books).

These "books" display a colorful, smoothly

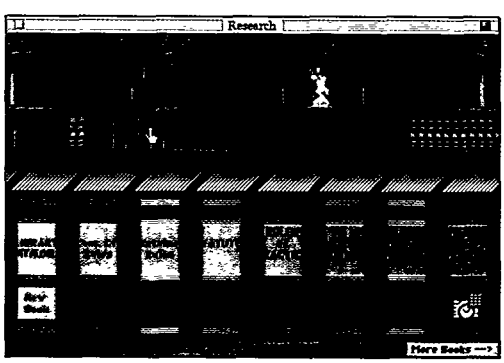


Figure 5
Gavel & Gown Software's Amicus Attorney. The user can access on-line legal services just as if they were using a more familiar library. Note the colors (oaks, dusty reds and greens, and gold) used to evoke a legal setting. All of the backgrounds used in this product come from an actual lawyer's office; their use in this product helps lower the intimidation factor with many new users, and adds to the status and enjoyment of those who are familiar with computer products.

Figure 6

A snapshot from Living Books' The Tortoise and the Hare. Here, the tortoise is listening to the birds sing. Note the microphone and musical instruments the birds are holding. Clicking on any of these birds (and many other areas on screen) causes unexpected things to happen.

animated interface that reads a story aloud to children and their parents (Figure 6). Further, the interface — an on-screen storybook page — uses obvious but unobtrusive modes (pages), direct manipulation of on-screen objects, and widely varying sounds to subtlely encourage the reader to explore the story and its background in a wonderfully engaging fashion. This is aided by providing a clear and bright visual presentation of the story elements, and an animated connection between the words (which highlight as the computer reads them) and characters in the story.

The pages are also filled with objects, characters, and regions that react in unexpected but delightful ways when activated with a mouse button click. These hidden rewards in the user interface have become known as "Easter eggs" and are beginning to show up in a variety of software products. Such devices are peripheral to the focus of most software; developers should use them judiciously, neither overusing nor underestimating the opportunity to captivate and encourage the user. In the case of the Living Books, the creators of this software seem to have taken the premise that their users (children of all ages) are intelligent individuals to be entertained but not pandered

to, and have used animation, sound, and Easter eggs to engage them emotionally, perceptually, and cognitively.

The world of the child learning to read may seem different from that of the radiologist or lawyer, but their needs are not dissimilar: All are examples of demanding users with little desire to learn computers but who can benefit from their use. These users respond well to products that provide clear value while engaging them on multiple levels and encouraging them to explore further.

The products described for these users have striking similarities as well. Each displays its functions and graphical data in their most natural and engaging forms; visual and auditory recognition are employed instead of making the user learn and recall arcane commands. Each product has a modular, task-oriented focus, though the "modes" are often smoothed over by a unifying interface metaphor. Each provides extensive direct-manipulation tools and environments. Finally, these products each provide access paths suited to both unfamiliar and seasoned users. This allows for quick starts and low-stress productivity gains as the user becomes more familiar with the product. While the products themselves are as different as the demanding users they serve, the user-centered approach to product design results in similar degrees of user acceptance for each.

Process

The process of designing for demanding users is necessarily a user-centered one. Designing instead for engineers (what technology can we build?), marketing (what is everyone else doing?), or management (what will make the stock go up next quarter?) may result in short-term or niche-based sales, but is unlikely to capture the minds of truly demanding users. Hallmarks of user-centered design include an iterative process involving

- early product and user analyses
- iterative design and prototyping
- and formative testing

as shown in the previous examples and by many authors [1, 2, 4]. There is no single user-centered design process that is accepted across the software industry. However, based on my experience, some principles and activities involved in this process are becoming clear.

At the beginning of the user-centered design process, all those involved in the development of a successful product must agree on the product's definition — one based on careful analysis of the target users, their task flow, informational requirements, and physical and organizational environments. Resisting the temptation to base the product definition on second-hand user knowledge, marketing fads, or the latest technological gimmickry will prevent later problems in design and deployment.

The encapsulation of the early product and user analyses comes in the form of a simple, clear mission-like statement backed up by usability goals and task-flow diagrams. The simple product definition acts as a touchstone during development so that "creeping featureism" is avoided. Usability goals help team members prioritize their work and know how different designs perform against user expectations. Task-flow diagrams (Figure 7) are the precursor of the more familiar feature lists and functional specifications and are close cousins to state-transition diagrams; they provide a map of the users' work showing the natural progression from one action to another in pursuit of the final goal. To facilitate team understanding and solid software architecture, these plans are best created in a hierarchical fashion (Figure 8).

Once the team agrees on the task focus and usability measures for the product, traditional software development processes come into play. This way, features and functional specifications

The world of the child learning to read may seem different from that of the radiologist or lawyer, but their needs are not dissimilar . . .

High-level tasks and transitions for the Physician's Review Station

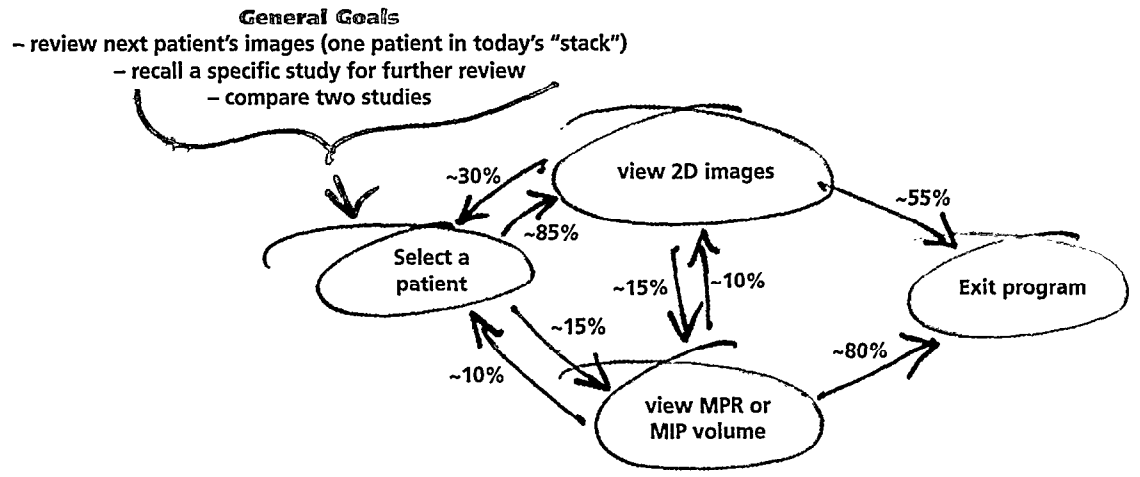


Figure 7
An example task-flow diagram. These start out simple, with the major states and activities highlighted, and gradually become more complex as more tasks are included.

emerge based on the supported tasks — not the other way around — with the added benefit of easy code modularization based on natural task groupings. The early agreement on tasks and matching features eliminates expensive arguments, course changes, and rework. It also highlights high-risk areas, thus enabling the team to pay close attention to scheduling, technological progress, or user-acceptance measures in specific areas.In designing the interface for the demanding user, information about the product itself must be made as "invisible" as possible. That is, the user interface must be more perceptually than cognitively significant [5], so the user can think about the task and not the tool. Constructing the features to fit the users' tasks — rather than hoping the users will change their work to fit the tools — accomplishes this. Using a few clear metaphors, graphics, animation, and icons greatly enhances this effort.

This commitment to clarity makes the product more transparent; it also provides the emotional engagement and sense of comfort that comes from knowing someone else has anticipated your needs. Deep levels of engagement are not often considered in developer-centered products, but those products designed for demanding users must be more than just feature-laden to be successful; their usability must be more than screen-deep.

While common tools must be represented clearly, not everything in the product needs to be equally obvious. Tucking advanced or occasionally used features out of sight where they do not get in the way of common usage removes many distractors, helping the user focus on the task at hand. However, avenues for exploration of the product must be clearly marked and provide for no-risk investigation at the user's own pace. In this way users grow their model of the product to fit their work, rather than reshaping their work to fit the product.

Finally, when system errors happen, recovering from them must be made as low-anxiety as possible. Demanding users do not tolerate programming laziness when things go wrong — uncommunicative error messages are merely a sign that similar development haste is likely to crop up elsewhere in the product, probably when they can least afford it.

As development proceeds beyond the product analyses and early designs, the user interface solidifies. First, the user interface designer creates task-based storyboards to try out various ideas with team members and a few users. These storyboards can be formalized on paper or even on-screen, but most begin as fragmented drawings on whiteboards, an invaluable tool for the fast generation of many ideas.

Once a few general ideas begin to win out over others, these show up as non-functioning mockups depicting primary metaphors, screens, and direct manipulation tools. Mockups are generated quickly, though they are more formal than storyboards. These take into account real-

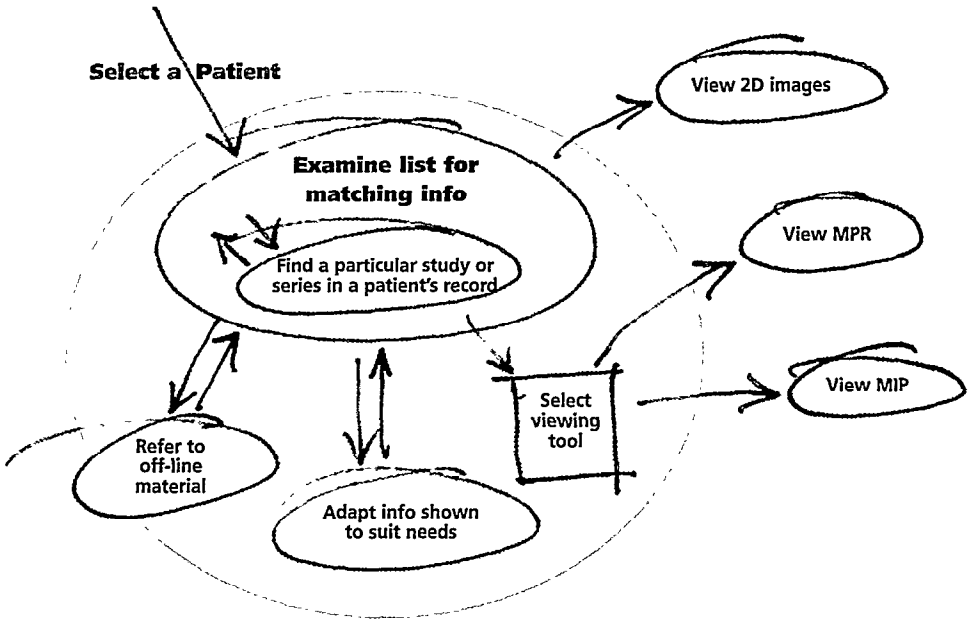


Figure 8
An example hierarchical task-flow diagram. Each task (bubble) at one level can be broken down into sub-tasks at a lower level. This can continue until going any further results in implementation rather than design decisions, though it is not typically necessary to go that far.

ities such as screen real estate and widget size and placement. Typically mockups are not interactive, and do not demonstrate the product-task flow. Thus, while mockups are useful for getting fast user feedback without investing a lot, they are problematic: Users (and others not intimately familiar with the ideas being shown) sometimes have difficulty commenting on an idea's utility when seeing it in an isolated, rough form. Since mockups are non-interactive, their creation requires little engineering support. However, the development team (and marketing, service, etc.) must be involved in their review — at least — to prevent divergence between the user interface and the underlying software modules.

Creating prototypes of user interface components comes next. These are often made from mockups, but differ in that they are more "real" and have both greater interactivity and specificity. A prototype typically demonstrates major task-related states and controls, including main screen sequences, icons, menus to be used, etc. They allow for formal user testing early enough in development to make a difference in the released product — without sacrificing the development schedule. The final user interface screens and components are often made from the prototypes as they consolidate, based on continuing user feedback. As the prototypes become more real, the development engineers become responsible for their creation and maintenance. Passing these smoothly from user interface designers to software engineers is an important milestone; cross-training members of both groups helps this along.

As the prototypes mature and the team integrates the product modules, the process shifts from design to evaluation. The borderline is not clear-cut except at common engineering milestones like the start of beta-testing. Unfortunately, if the team conducts no user evaluation until this point, there is rarely time to do more than a few token usability tests, or to implement more than a few (if any) of the changes recommended by this testing. It is also likely that you will find development resources have been wasted on unimportant details, or on "refining" a feature that was good enough months earlier.

To ensure successful product development, user testing takes place repeatedly as the product develops. These tests should be quick and informal when the product is itself at an informal or semi-defined stage, and increase in scope and formality as the product matures. Properly conducted, such formative testing has at worst a neutral effect on the development schedule, and can have a positive effect by eliminating expensive detours and rework. It is vital, however, to publicize the findings of usability tests to all those concerned with the product's development.

One of the problems with doing things better is that you don't get credit for mistakes you don't make: saving six months in a development

schedule by preemptively eliminating an unnecessary component is significant for any project, but such "savings" rarely show up on the bottom line and are quickly forgotten. Increasing the visibility of iterative design changes thus helps obtain internal acceptance for user feedback as a valuable information source. It helps to remember too that any evaluation results that are not used in one version of a product will likely serve as advance analysis for the next.

Throughout the user-centered development process, some engineers are uncomfortable spending the time it takes to learn from users when they could be coding (doing "real work"). They forget that someone will analyze and evaluate their designs — if no one else does, their users and competitors will. Demanding users do not have attentional and cognitive resources to squander figuring out subtleties in a new user interface, and cannot be "forced" into doing so (i.e., by enforced product use and long learning curves). Since demanding users are quick to take their business elsewhere, ignoring the user-centered process increases the risk of slipped schedules, missed budgets, poor sales, and a shortened product life — a potential disaster for the manufacturer.

The issues encountered when designing for demanding users can serve as advance models for product designers in general. As more people become aware of their technological choices and as our industry matures, captive users are disappearing — we are all becoming demanding users. Except for certain small deep-niche markets, the days are gone when manufacturers could rely on the users' affinity for computers, their willingness to spend weeks at the bottom of the productivity and learning curve, their lack of alternative choices, or their lack of buying authority to prop up their sales. When we regard all potential customers as demanding users and increase the maturity and user-centered focus of our development process, we gain greater user satisfaction, increased sales, and increased implementation of the user-centered design process.

References

[1] Dayton, T. et al Skills Needed by User-Centered Design Practitioners in Real Software Development Environments: Report on the CHI'92 Workshop. SIGCHI Bulletin 25 3 (July 1993)

[2] Gould, J.D. and Lewis, C. Designing for Usability: Key Principles and What Designers Think. Communi. ACM 28 3 (March 1985), 300-311

[3] Hix, D. and Hartson, R. Developing User Interfaces. Wiley, New York, 1993

[4] Nielsen, J. Usability Engineering. Academic Press, Boston, 1993

[5] Robertson, G.G., Mackinlay, J.D., and Card, S.K. Cone Trees: Animated 3D Visualizations of Hierarchical Information. In Proceedings of CHI'91. ACM, New York, 1991, 189-194