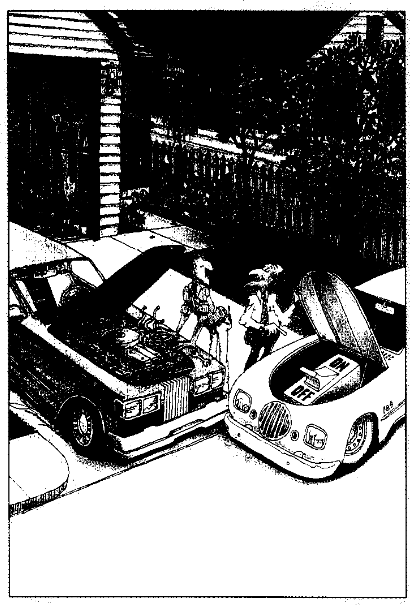


The Usability Engineering Life Cycle

Jakob Nielsen, Bellcore

A usability engineering process to ensure good user interfaces includes elements to be considered before the design, during the design, and after field installation of a software product.

Computer user interfaces have become more important with the increase in number of users and applications. The personal-computer revolution and falling hardware prices made computers available to ever broader groups of people who use computers for a larger variety of tasks. Initially, when computers were used by only a few people performing specialized tasks, it made some sense to require a high degree of user expertise. Also, because computers were so expensive, it was not unreasonable to let users suffer a little in favor of computational efficiency. Now, however, it pays to dedicate a large portion of computational resources — CPU cycles, memory use, communication bandwidth, screen space, and development effort — exclusively to making life easier for the user.

Users are becoming less willing to put up with difficult or uncomfortable interfaces since experience with some current interfaces has shown them that software can indeed be easy to learn and pleasant to use. In an unpublished study from 1990, Tim Frank Andersen of the Technical University of Denmark read 70 reviews of software products in various personal computer magazines and counted 784 comments on the usability of the reviewed software. This is an average of 11.2 usability comments per software review. Many of these comments were fairly superficial, but their sheer number indicates the importance of usability to today's users.

High usability is thus desirable, but it does not magically appear just because we want it. To ensure the usability of interactive computer products, we must actively include usability concerns in the software development process. Of course, nobody deliberately sets out to design an unusable interface, but only a systematic usability effort using established methods can qualify as usability engineering. Good intentions are not enough.

This article presents a practical usability engineering process that can easily be incorporated into the product development process as steps to be taken in roughly chronological order. Because the article considers the life cycle, several of the steps are iterative and some may overlap. The actions needed to ensure usability form the usability process. The model presented should therefore be seen as advice about what to include in the design and implementation process. In this context, I am not giving advice about the properties of the product of this process. Many such guidelines exist, and studying and applying selected guidelines is one of the recommended steps.

0. Consider the larger context
1. Know the user
   - Individual user characteristics
   - The user's current task
   - Functional analysis
   - Evolution of the user
2. Competitive analysis
3. Setting usability goals
4. Participatory design
5. Coordinated design of the total interface
   - Standards
   - Product identity
6. Guidelines and heuristic analysis
7. Prototyping
8. Empirical testing
9. Iterative design
   - Capture the design rationale
10. Collect feedback from field use

$\}$ Apply metamethods throughout

Prioritize the usability methods

Figure 1. Elements of the usability engineering model.

Usability engineering model

The model presented here is a modified and extended version of Gould and Lewis' "golden rules": early focus on users, user participation in the design, coordination of the different parts of the user interface, empirical user testing, and iterative revision of designs based on the test results.[1] Further inspiration and modifications came from work on usability engineering.[2,3]

Figure 1 lists the elements in the complete usability engineering model. However, the effort can be successful without including every refinement. (See the final section of this article for a prioritization of methods under varying levels of resource constraints.)

The most basic elements in the usability engineering model are empirical user testing and prototyping, combined with iterative design. Because it's nearly impossible to design a user interface right the first time, we need to test, prototype, and plan for modification by using iterative design. Under typical resource constraints, modifications will be feasible only in the prototyping stage. It is much too expensive to change a completely implemented product, especially if testing reveals the need for fundamental changes in the interface structure.

Product development context

The following sections present usability activities for three main phases of a software project: before, during, and after product design and implementation. The constraints of the print medium necessitate a sequential presentation of these usability activities, even though they should really be applied iteratively in the manner of Boehm's spiral model of the software process.[4]

Software development and user interface design are both part of a broader corporate product-development context in which one-shot projects are fairly rare. Usability should apply to the development of entire product families and extended projects where products are released in several versions over time.

This broader context strengthens the arguments for allocating substantial usability engineering resources as early as possible. Design decisions made for early products have ripple effects because subsequent products and versions must be backward compatible. Consequently, some usability engineering specialists believe that "human factors involvement with a particular product may ultimately have its greatest impact on future product releases."[5] Of course, having to plan for future versions is also a compelling reason to follow the initial product release with field studies of its actual use.

The term *life cycle* is normally defined (IEEE standard 100-1988) as starting when a software product is conceived and ending when the product is no longer available for use. The usability engineering life cycle extends beyond this period because of the impact of usability decisions on future products and their life cycles. We must consider not just how an interface design meets current needs but also whether it conflicts with skills users have acquired from previous interfaces and whether it seems flexible enough to be extended for future interfaces.

Predesign stage

The first stage of the usability life cycle aims at understanding the target user population and user tasks. We can make valid design decisions and appropriate trade-offs only when we understand these factors, so gathering this information should be the first priority.

We should not rush into design. The least expensive way for usability activities to influence a product is to do as much as possible before design is started. Then it will not be necessary to change the design to comply with the usability recommendations, and it may be possible to avoid developing unnecessary features.

Several of the predesign usability activities might be considered part of a market research or product planning process and be performed by marketing groups. However, traditional market research does not use all the needed usability design methods, and the results are often poorly communicated to developers. But there should be no need for duplicate efforts if management successfully integrates usability and marketing activities. An outcome of such integration could be the consideration of product usability attributes as features to be used by marketing to differentiate the product.

Know the user. The first step in the usability process is to study the product's intended users. As a minimum, developers should visit a customer site to gain a feel for how the product will be used. Individual user differences and variability in tasks are the two factors with the largest impact on usability, so they need to be studied carefully. Developers should also keep in mind that users often include installers, maintainers, system administrators, and other support staff, in addition to the people who sit at the keyboard. The concept of "user" should be defined to include everyone whose work is affected by the product.

Individual user characteristics. We should know the type of people who will be using the system. In some situations it's possible to identify the users as specific individuals — for example, when the product is going to be used in a specific department in a particular company. For products with more widely scattered users, it's possible to visit only a few, representative customers. Alternatively, the products might be aimed toward the entire population or a very large subset.

By knowing the users' work experience, educational level, age, previous computer experience, and so on, we can anticipate their learning difficulties to some extent and better set appropriate limits for user interface complexity. Certainly, we also need to know the users' reading and language skills. For example, very young children with no reading ability require a nontextual interface.

The users' work environment and social context are also important. As a simple example, the use of audible alarms, "beeps," or more elaborate sound effects may not be appropriate for users in open office environments. In a field interview I conducted, a secretary insisted on the ability to shut off the beep; she feared that others would think she was stupid if her computer beeped at her all the time.

User differences and task variability are the two factors with the largest impact on usability. Study them carefully.

A great deal of the information needed to characterize individual users may come from market analysis or as a side benefit of observational studies conducted for task analysis. We can also collect such information directly through questionnaires or interviews. In any case, it's best not to rely totally on written information. New insights are almost always achieved by observing actual users in their own working environments.

The user's current task. A task analysis is extremely important as early input to the system design. The users' overall goals should be studied, as well as how they currently approach the task, what their information needs are, and how they deal with exceptional circumstances or emergencies. For example, systematic observation of users talking to their clients may reveal input and output needs for a transactions processing system. The users' model of the task should also be identified, because it can be used as a source for metaphors for the user interface. Also, we should seek out and observe especially effective users and user strategies and "work arounds" as hints of what a new system could support. Finally, we should try to identify the weaknesses of the current situation: points where users fail to achieve goals, spend excessive time, or are made uncomfortable. These weaknesses present opportunities for product improvements.

Functional analysis. A new computer system should not propagate suboptimal methods that may have been instituted because of limitations in previous technologies. Therefore, we should not just analyze the way users currently do the task, but also the underlying functional reason for the task: What is it that *really* needs to be done? What are merely surface procedures that can, and perhaps should, be changed? At the same time, there is a limit to how drastically we can change the users' task, so the functional analysis should be coordinated with the task analysis.

Evolution of the user. Users do not stay the same. Using the system changes the users, and as they change, they use the system in new ways that are impossible to forecast completely. Users will always discover new uses for computer systems, and a flexible design stands a better chance of supporting these new uses. Thus, we should make an educated guess about future users and uses, based on our knowledge about how other users have changed in the past. One way of getting such knowledge is through the postdeployment field studies recommended as the last step of the usability process.

A typical change is that users eventually become experts and want interaction shortcuts (sometimes called accelerators). It is important to avoid designing only for the way users will use the system in the first short period after its release.

Competitive analysis. Prototyping is an important part of the usability process, and existing — perhaps competing — products are often the best prototypes of our own product. We should analyze existing products heuristically according to established usability guidelines (discussed in the next section) and perform empirical user tests with these products.

A competing product is already fully implemented and can therefore be tested very easily. Also, its developers often put a reasonable effort into *their* development process, so the competing product may work fairly well. User testing with an existing product can be more realistic than a test of other prototypes. By having users perform real tasks on the competing system, we can learn how well its functionality and interaction techniques support the kinds of tasks we expect the planned new product to support.

If several competing products are available, we can do a comparative analysis of the different approaches to the user interface design issues we're studying. This will provide ideas for the new design and ad hoc guidelines for approaches that seem to work and others that should be avoided.

A competitive analysis does not imply using other people's copyrighted designs. We hope to do better as a result of analyzing their strengths and weaknesses.

Setting usability goals. The five main usability characteristics are

- learnability,
- efficiency of use once the system has been learned,
- ability of infrequent users to return to the system without having to learn it all over,
- frequency and seriousness of user errors, and
- subjective user satisfaction.

Obviously, these five parameters cannot be given equal priority in any single design, and clear priorities, based on the analysis of the users and their tasks, should be set. For example, high learnability would be especially important if new employees were constantly being brought in on a temporary basis. The ability of infrequent users to easily return to the system would be especially important for a system-reconfiguration utility used once every three or four months. In many cases, however, the five usability characteristics tend to be positively correlated rather than in conflict, so getting good results on all of them is normally a reasonable goal.

Usability goals should be specified in more detail than the five general usability parameters, and doing so is an important part of the usability process. Not all the goals we specify have to be measured, but just knowing (and agreeing on) goals helps clarify the design process. Important goals should be specified in more detail than less important goals, and some goals should be specified in sufficient detail to allow empirical measurement of the degree to which the product achieves these goals.

The development team should participate in defining the goals so that its members won't see usability goals as outside interference with their project. Developers who buy into the goals will be more motivated to fulfill them.

In specifying usability goals, several different levels of the attributes can be listed.[2] The most important may be the *worst acceptable level*, because it indicates that the product would be of no use if that level of usability is not achieved. Furthermore, we should specify the *planned usability level*. Additional levels are the *current level* of usability observed in competitive systems, or in whatever methods users currently use to perform the task, and the *best possible level* for the usability attribute.

Consistency, an important usability characteristic, should apply across all media forming the total user interface.

As an example, consider the goal for user errors in a system for electronic submission of expense accounts in a company where 10 percent of the paper forms have contained errors. That 10 percent constitutes the current level and zero errors the best possible level. Because expense report errors are not catastrophic, it may not be reasonable to specify zero errors as the planned level, but 2 percent might be reasonable. Six percent may be the worst acceptable level since it may not be worthwhile to change the reporting procedures unless a significant improvement can be obtained. Of course, usability goals should be set in a trade-off with any further system attributes, so the worst acceptable user error level might be 15 percent if significant cost savings were expected from the electronic processing of the documents.

Design stage

After completion of the predesign stages, several steps should be followed in carrying out the design process. In most cases, we cannot follow a strict order of activities because of the fundamental need for an iterative design approach that gradually refines the user interface in several passes through the design process.

The main objective of the design phase is to arrive at a usable implementation that can be released. For this to happen, we must meet two further objectives: getting a concrete embodiment of the design in a prototype that follows established usability principles, and empirically verifying the design with real users to ensure that it meets their needs.

Participatory design. Even if we follow the advice to "know the user" before the design phase, we still cannot know the user completely enough to answer all issues that will come up. Instead of guessing, designers should have access to a pool of representative users after the start of the design phase. Furthermore, users often raise questions that the development team has not even dreamed of asking. This is especially true with respect to potential mismatches between the users' actual task and the developers' model of the task. Therefore, users should be involved in the design process through regular meetings between designers and users.

Users are not designers, so we should not expect them to come up with design ideas from scratch. However, they are very good at reacting to concrete designs that they do not like or that will not work in practice. To get the full benefit of user involvement, we must present the suggested system designs in a form the user can understand. Instead of voluminous system specifications, use concrete and visible designs, preferably in the form of prototypes. In the early stages of the design, when functional prototypes are not yet available, paper mock-ups or a few screen designs can prompt user discussion. Even simple, guided discussion can elicit ideas.

It is important to reach the people who will actually use the system, not just their managers. For example, in developing a computer-aided instruction system, we need access both the teachers and the students. However, teachers have an authoritative position with respect to the students, so we should talk to members of each group separately. In any case, it is probably very difficult to involve young children directly in the design. For systems to be used by young children, we have to rely mostly on empirical testing, not on participatory design.

Coordinated design. Consistency is one of the most important usability characteristics.[6] Consistency should apply across the different media that form the total user interface, including not just the application screens but also the documentation, the on-line help system, and any on-line or videotaped tutorials. Also, consistency is not measured just at some specific point in time. It should apply over successive releases of a product so that new releases are consistent with their predecessors. Despite its general desirability, consistency sometimes conflicts with other desirable usability characteristics. Some flexibility is necessary to avoid forcing a bad design on users for the sake of consistency.

To achieve consistency of the total interface, a centralized authority for each development project should coordinate the various aspects of the interface. Typically this coordinator can be a single person, but on very large projects or for corporate-wide standards, a committee may be more appropriate.

In addition to formal coordination activities, it helps to have a shared culture in the development groups — that is, a common understanding of what the user interface should be like. Many aspects of user interface design (especially the dynamics) are hard to specify in written documents but can be fairly easily understood from looking at existing products following a given interface style. Actually, prototyping also helps achieve consistency, since the prototype is an early statement of the kind of interface the project is aiming toward. An explicit instance of parts of the design makes the design details more salient for developers and encourages them to follow similar principles in subsequent design activities.

Consistency can be increased through technological means such as code sharing or a constraining development environment. When several products use the same code for parts of their user interfaces, those parts of the interfaces will automatically be consistent. Even if identical code cannot be used, we can provide development tools and libraries that encourage user interface consistency by making it easiest for developers to implement interfaces that follow given guidelines.

Standards. Interface standards are currently a popular approach to achieving consistency. A standard can be a widely followed de facto standard such as those promoted by several vendors and window systems, or it can be an in-house standard. The advantage of a de facto standard is that it ensures product consistency with a large set of products developed by other companies. The advantage of in-house standards is that they can be tailored to the needs of the special kind of application normally developed by a specific company. Both kinds of standards may increase the reuse of code and documentation.

Use simple and natural dialogue
Speak the user's language
Minimize user memory load
Be consistent
Provide feedback
Provide clearly marked exits
Provide shortcuts
Provide good error messages
Prevent errors

Figure 2. Nine usability heuristics.[3,7]

Formal international standards for some aspects of user interfaces are apt to be promulgated within a few years. However, such standards are not likely to constrain user interfaces sufficiently to form the only basis for consistency. It is possible to adopt a general standard and supplement it with a set of house rules for various design details such as graphical look and choice of vocabulary. We could, of course, do with just the house style guide and avoid the larger standards, but that would risk longer training time for new employees accustomed to the main interface standards.

Standards and guidelines differ in that a standard specifies how the interface should appear to the user, whereas a set of guidelines provides advice about its usability characteristics. (Guidelines are discussed in the next section.) A given standard should follow most of the traditional usability guidelines so that interfaces designed according to the standard will be as usable as possible. For example, a guideline may state that users should always have an easy way out from any undesired system state. One standard might instantiate that general guideline by specifying that an Undo command should always be available and shown as an icon at the top right of the screen. Another standard might follow the same guideline by returning to the previous system state whenever the user hits the Escape key.

Product identity. A product identity statement is a high-level description of what kind of "thing" the product is. It specifies the project's overall goals: what the product is supposed to be good for, who is going to use it, and what other products it will be used with. The product identity statement can help coordinate the design because it is a short document known by all members of the development team.

The project manager should write and review the product identity document before the start of the design process and then modify it sparingly.

Guidelines and heuristic analysis. Guidelines list well-known principles for user interface design that should be followed in the development project. In any given project, several different levels of guidelines should be used:

- *general guidelines* applicable to all user interfaces,
- *category-specific guidelines* for the kind of system being developed (for example, guidelines for window-based administrative data processing or for voice interfaces accessed through telephone keypads), and
- *product-specific guidelines* for the individual product.

General guidelines are often published in technical journals or books. Some guideline documents contain thousands of guidelines, while others are less voluminous, comprising tens or hundreds of rules. A short set of guidelines (such as in Figure 2) is suitable for design inspiration or as a checklist in heuristic evaluation, while a large set of guidelines can serve as a reference for answering specific design questions.

Since good published guidelines are available, in-house development of general usability guidelines is seldom worthwhile. Also, a complete set of category-specific guidelines can often be found in the published literature, but we might have to adjust them somewhat to fit the precise kind of products being developed. Finally, the product-specific guidelines must, by definition, be developed for each project on the basis of observations of what does or doesn't work in the testing of competitive products and initial prototypes.

One general usability principle is to tell the user what is going on by providing feedback. For example, a file system could indicate that a file has been deleted by removing its name or icon from a list of current files. A category-specific principle for hypertext systems (see the sidebar) is to provide users with a sense of location in the information space but avoid showing a complete overview diagram of all nodes and links if the document is too large. For a large electronic handbook with a strict chapter-section-subsection hierarchy, a product-specific guideline could then provide feedback on the location with a fish-eye view showing the current specific location against an indented list of the subheading hierarchy.

It is possible to perform heuristic evaluation on the basis of the guidelines.[7] This is done by going through the interface design and determining whether each of its elements follows each of the guidelines established for the project. The actual activity can take the form of formal walk-throughs with elaborate checklists, or it can be performed more casually. But even the most casual heuristic evaluation should be based on some usability guidelines, not just personal opinion. The advantage of heuristic evaluation is that it can be done in the very early design stages because it does not require a running system. But I stress that heuristic evaluation should only supplement empirical testing.

Usability guidelines often contain apparent contradictions that can be difficult to resolve for people who are not usability specialists. To make appropriate trade-offs, we need to understand the spirit behind the guidelines, and this requires understanding higher level usability principles such as consistency. The same is true for applying results from the human factors research literature, since we cannot expect the literature to contain explicit design decisions. Guidelines that contain lists of advantages and disadvantages of various design approaches can also help us make trade-offs, but they are, again, difficult to apply for those who are not usability specialists.

In general, the project should have access to a usability expert to help re-

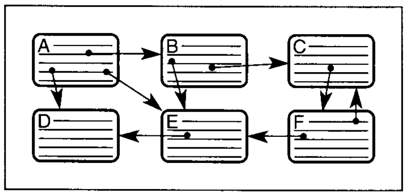
Hypertext

Hypertext[1,2] interconnects related pieces of information in a computer so that the user can move to new locations in the information space by following the connecting links. The information is normally divided into units, which are often displayed in separate windows on the screen. These units are called nodes because the entire hypertext information space forms a graph structure.

Navigation. A major issue in the design of hypertext systems is how to support the users' navigation through the information space. In the example in the figure, users might jump directly from node A to node D, or they may take the path A→E→D or even the path A→B→C→F→E→D. Because of this great freedom in moving about, users can easily get confused about where they are, where they came from, and where they can go. To alleviate these problems, hypertext systems often include some kind of overview diagram somewhat like the figure, as well as a history facility listing the previously visited nodes.

Fish-eye views. Fish-eye views[3] increase users' sense of location in an information space by showing great detail for those parts of the space close to the user's current location of interest and gradually diminishing amounts of detail for those parts progressively farther away. The use of fish-eye views therefore requires two properties of the information space: It should be possible to estimate the distance between a given location and the user's current focus of interest, and it should be possible to display the information at several levels of detail. This is especially easy to do in a hierarchically structured electronic book in which distant chapters can be displayed by their chapter heading only. Closer chapters can show additional levels of section and subsection headings.

Applications. The most obvious hypertext application is probably on-line manuals. Users of a software package will already be at their computers when they want to look up something in the manual. For many other applications, the need to be at a computer is somewhat of a disadvantage compared with the use of printed books — at least given current computer hardware. Even so, hypertext has many diverse applications, including museum and tourist information, electronic encyclopedias, teaching classic Greek literature, legal information for patent lawyers, auditing, brainstorm support, programming environments, and games.



Hypertext graph structure.

Design rationale. Hypertext can capture the rationale for user interface design decisions in a network of interrelated design issues and arguments pro and con. For example, an issue in the design of a paint program could be whether to present the various colors as a permanently visible palette or a pop-up menu, or to have the user explicitly type in the color mixture as percentages of red, green, and blue. There would be one hypertext node for the overall issue of color selection, with links to separate nodes for each possible solution. These nodes would have small sketches of how each interface design would look, and they would be linked to further nodes with the results of any user testing or heuristic evaluation. Because user interface decisions affect one another, there would also be links to, say, nodes about the use of screen space: The possible decision to use a permanently visible palette would diminish the space available for other interface elements and the primary drawing window.

References

1. J. Conklin, "Hypertext: An Introduction and Survey," *Computer*, Vol. 20, No. 9, Sept. 1987, pp. 17-41.
2. J. Nielsen, *Hypertext and Hypermedia*, Academic Press, San Diego, Calif., 1990.
3. G.W. Furnas, "Generalized Fish-Eye Views," *Proc. ACM CHI 86*, ACM, New York, 1986, pp. 16-23.

solve contradictory guidelines and to help in heuristic evaluation. Although different usability experts may give different advice, this does not necessarily imply that at least one of them is wrong. There are so many degrees of freedom in user interface design that several solutions may be more or less equally reasonable.

Prototyping. Experimental prototyping is highly recommended for the early stages in the design process. For software systems, we should "plan to throw one away"[8] because the first design will never be good enough. If anything, this advice is truer of interfaces than of other components because of the greater difficulty of predicting their flaws. It is less expensive to throw away a prototype than a completely implemented, fully functional system.

In traditional software engineering models, most of the development time is devoted to refining various intermediate work products, and executable programs are produced at the last possible moment. A problem with this approach is that there is no user interface to test with real users until this last possible moment, because the "intermediate work products" do not explicitly separate the user interface in a prototype with which users can interact. Experience also shows that it is not advisable to involve the users in the design process by showing them abstract specifications documents; they do not understand documents nearly as well as concrete prototypes.

It's best to postpone final implementation until late in the development process so that it can be done on the basis of experiences with the prototypes. The early prototypes can be quite primitive (for example, paper mock-ups), whereas later prototypes can be progressively closer to the final product. Often we can gain substantial insights from low-fidelity prototypes,[9] so it is probably better to prototype early and prototype often than to put an extensive effort into a single, elaborate, and (too) late prototype. Even computerized prototypes may be implemented faster and cheaper in systems other than the eventual delivery platform, and implementation time can sometimes be reduced by "cheating" on the algorithms to make them ignore the special cases that often take an inordinate amount of programming effort.

Prototype early and prototype often to avoid putting extensive effort into a single, elaborate, (too) late prototype.

Empirical testing. The most basic recommendation for empirical testing is simply to do it. The benefits of doing some user testing versus doing no user testing are much greater then the differential benefits of various approaches to testing.

There are two basic forms of empirical testing:

(1) Testing a more or less finished interface to check whether the usability goals have been achieved. This kind of testing implies doing some form of quantitative measurement.

(2) Formative evaluation of a system still being designed to see which aspects of the user interface work and which cause usability problems. This kind of testing is often best done using qualitative methods. At this stage, it is more important to know why the interface is wrong than how much it is wrong.

In both cases, it is important to have the test users perform tasks representative of the eventual use indicated by the predesign task analysis.

Some of the more common test methods are

• Thinking aloud (see the sidebar).
• Constructive interaction (sometimes called codiscovery learning). In this method, two users work together to perform the test task. This approach is better than traditional thinking aloud when the test subjects are children, because verbalizing comes more naturally in a two-user setting. Even adults often find this method more natural, but it does have the obvious disadvantage of requiring twice as many test users.
• Attitude questionnaires in which users rate designs on, say, a 1 to 5 scale.
• Tests of the user's level of knowledge (or other user skills) before and after using the system.
• Automatic computer logging of user actions. Later analysis can determine, say, that a certain error message is issued so frequently that the "prevent errors" usability guideline should be applied to reduce the probability of the corresponding error situation.
• Observation of users working in their natural environment with their own tasks.
• Observation of users working on a set of representative standard tasks.

The last two methods are especially suited for follow-up tests of released products or for tests of prototypes that are complete enough to allow users to do real work.

From the results of the empirical tests, we obtain a list of usability problems in the test version, as well as hints for features to support successful user strategies. It's not feasible to solve all the problems, so we prioritize them. The ranking should be based on experimental data about the impact of the problems on user performance (for example, how many people will experience the problem and how much time each of them will waste because of it).

But sometimes we must rely on intuition only. In some cases, solving a problem may make the interface worse for those users who do not experience the problem. Then a trade-off analysis is necessary to determine whether to keep or change the interface, based on a frequency analysis of how many users will have the problem compared with how many will suffer because of the proposed solution.

The time and expense needed to fix a particular problem is also a factor in determining priorities. Often, usability problems can be fixed by changing the wording of a menu item or an error message. Other design fixes may involve fundamental changes to the software (which is why they should be discovered as early as possible) and will be implemented only if they are judged to affect usability significantly.

Iterative design. On the basis of the usability problems and opportunities disclosed by the empirical testing, we can produce a new version of the interface. Some testing methods, such as thinking aloud, provide sufficient insight into the problems to suggest specific changes to the interface. In other cases, alternative potential solutions need to be designed solely on the basis of usability guidelines, and it may be necessary to test several possible solutions before making a decision. Familiarity with the design options, insight gained from watching users, creativity, and luck are all needed at this point.

Some of the changes we make to solve certain usability problems may fail to solve the problems or even introduce new ones. This is another reason for doing iterative design and evaluation.

Retesting. Additional usability problems will likely appear in repeated tests after the most blatant problems have been corrected. There is no need to test initial designs comprehensively since they will be changed anyway. We should change and retest the user interface as soon as we detect and understand a usability problem so that we can find the remaining problems that were masked by the initial glaring problems.

During the iterative design process it may not be feasible to test each successive version with actual users. The iterations are a good way to evaluate design ideas simply by trying them out in a concrete design. We can then subject the design to heuristic analysis and show it to usability experts and consultants, or discuss it with expert users (or teachers, in the case of learning systems).

We should not "waste" users by performing elaborate tests of every single design idea. Test subjects are normally hard to come by and should be conserved for the testing of major iterations. Also, users get "worn out" as appropriate test subjects. As they get more experience with the system, they stop being representative of novice users seeing the design for the first time. Users who have been involved in participatory design are especially inappropriate as test subjects because they will be biased.

Design rationale. The rationale for the various user interface design decisions can be made explicit and recorded either in traditional written form or in a hypertext structure such as gIBIS.[10] Having access to such an audit trail is important during iterative development and during development of any future product releases. Because we will often have to change the interface, we should know the reasons for the original design to avoid sacrificing important usability principles to attain a minor objective. Furthermore, the design rationale can help in maintaining user interface consistency across successive product versions.

Thinking aloud

Thinking aloud is a commonly recommended method for user testing that can be used for almost any system, so I present it in somewhat more detail than the other methods mentioned in this article. Many of the recommendations also apply to other forms of user testing.

Basically, a thinking-aloud test involves having a test subject use the system while continuously thinking out loud. While verbalizing their thoughts, the test users reveal their view of the computer system, and this lets us identify their major misconceptions. We get a very direct understanding of what parts of the dialogue cause the most problems, because the thinking-aloud method shows how users interpret each interface item.

The thinking-aloud method has traditionally been used as a psychological research method,[1] but it is increasingly being used for the practical evaluation of human-computer interfaces.[2] Studies have shown[3] that computer professionals can use the thinking-aloud method to good effect. Often, it is enough to run a fairly small number of test users (4±1) to find most usability problems. The main disadvantage of the thinking-aloud method is that time measurements will not be representative of real usage because verbalizing thoughts and answering questions will slow down the user.

Test tasks. The experimenter must prepare a set of tasks for the test user, since the experience of using an application differs significantly depending on whether the user is just fooling around or is trying to achieve a set goal. We are normally interested in testing the usability of software to achieve a goal. The tasks should be chosen to reflect typical, serious usage situations.

Running the test. Unfortunately, it is rather unnatural for most people to continuously verbalize their thoughts. Therefore, the experimenter often needs to prompt the test user with questions like "What are you thinking now?" or "How do you interpret this error message?" The experimenter should not answer any questions asked by the user because the test aims at assessing how easy the user interface is to use without outside help. To increase the test user's confidence, the experimenter should ensure an early success experience by having the very first assignment be extremely easy.

Ethical considerations. Test results from individual users should be kept confidential. It would immediately ruin any constructive atmosphere if, say, the users' manager was to use test scores to assess their computer skills. Even so, test users will always feel as though they are taking an exam, and they will feel stupid whenever they make mistakes. To reduce the unpleasantness of the test, experimenters should inform the test users that they are not testing the users but that they are testing a preliminary software design that is bound to have some problems. Finally, test subjects should be allowed to discontinue the test at any time if they find it too unpleasant (their natural pride will ensure that they will almost never do so).

References

1. K.A. Ericsson and H.A. Simon, *Protocol Analysis: Verbal Reports as Data*, MIT Press, Cambridge, Mass., 1984.

2. S. Denning et al., "The Value of Thinking-Aloud Protocols in Industry: A Case Study at Microsoft Corporation," *Proc. Human Factors Soc. 34th Ann. Meeting*, Human Factors Society, Santa Monica, Calif., 1990, pp. 1285-1289.

3. J. Nielsen, "Evaluating the Thinking-Aloud Technique for Use by Computer Scientists," in *Advances in Human-Computer Interaction*, Vol. 3, H.R. Hartson and D. Hix, eds., Ablex, Norwood, N.J., 1992, pp. 75-88.

Postdesign stage

The main objective of usability work after product release is to gather data for the next version and for new future products. In the same way that existing and competing products were the best prototypes for the product in the initial competitive analysis phase, a newly released product can be viewed as a prototype of future products, and in most cases it is certainly the prototype of its own next release. Therefore, we must not end the usability process with the initial release of the product to the marketplace; we need to conduct follow-up studies of product use in the field. Such studies assess how real users use the interface for naturally occurring tasks in their real-world working environments and can lead to insights not readily available from laboratory studies.

A simple way to obtain feedback from users of installed products is to log user calls to hot lines or other product-support structures. It is important to go beyond simply recording immediate complaints and classify the problems to determine patterns and likely root causes. However, this only provides feedback about problems. To learn about the system's positive aspects — and its new, unexpected uses — we need to visit real users in their everyday work environments. Also, logging user sessions with the installed system is a good way to obtain field data on system use.

Finally, economic data on the system's impact on the quality and cost of the users' work product and quality of their work life are very important. We can gather these data through surveys, supervisor opinions, statistics for absenteeism, and so on. These data should be compared with similar data collected before the introduction of the system.

Life cycle model summary

Look again at Figure 1, the outline of activities recommended in our usability engineering process, and note that some of the recommended methods are not really single "steps" but should be used throughout the process.

Costs. Obviously, there is some cost associated with following the recommended usability engineering process, even though we can significantly reduce these costs by concentrating on a subset of the methods. But usability is not just a cost item in a development project, even though the "benefits" side of the cost-benefit trade-off is articulated with comparatively poor precision and evidence in the usability literature.[11] (One documented case study shows savings of $41,700 in reduced training time for one small in-house product as a result of a $20,700 usability effort.[12])

The financial payoff associated with users' ability to learn the product faster and work more productively is spread over the entire period of product use and is therefore hard to measure precisely. And these benefits are certainly not explicitly visible during development. A major benefit for the development team itself, however, is the time saved in not implementing features that the usability analysis shows are not needed by users. Furthermore, in many situations usability is a major marketing consideration, and an otherwise acceptable product will fail completely if it is not perceived as usable by customers.

In any case, the cost-benefit relation may be drastically changed when considered in the context of the entire product life cycle rather than a single-release context. This is especially true if we take the even broader corporate perspective of multiproduct development.[5] A benefit of early usability efforts may manifest itself in fewer customer modification requests if users' needs are matched better from the start. Another benefit may be the ability (and willingness) of users to learn and adopt additional products if they are easier to use.

Metamethods. To ensure the successful application of the usability engineering methods discussed here, it is important to supplement each with the following "metamethods" (methods that apply to methods):

- Write down an *explicit plan* for what to do when using the method. For example, a plan for empirical user testing should include information about how many users to test, what kind of users to test (and how to get them), what test tasks these users will be asked to perform (which itself should be based on task analysis and user observation), and a time schedule for the studies.
- Subject this plan to an *independent review* by a person who is not otherwise on the development team and who can critique it from a fresh perspective. Of course, this person should be experienced in usability engineering.
- Perform a *pilot activity* by investing no more than about 10 percent of the total resources budgeted for the use of the method. Then revise the plan for the remaining 90 percent to fix the difficulties that invariably will be found during the pilot activity. In empirical user testing, users often misinterpret the original test tasks, so be sure that the main test focuses on system usability, not on the developers' ability to write readable test instructions.

As early as possible in the project, establish an *overall usability plan* listing the usability activities to be performed throughout the life cycle. Not all projects can afford to use all the methods, and the exact methods to use will depend on the project characteristics.

These metamethods may involve a little extra work up front, but they save work in the long term and ensure that our efforts are on the right track to increase usability, thereby reducing the risk of truly wasting the main effort.

Prioritizing usability methods

As a reality check on the practical applicability of the usability methods suggested above, I asked 13 usability engineers to complete a questionnaire. For each of the methods listed in Table 1, the questionnaire asked whether they had used it in their most recent development project and what impact they felt the method had on usability in general (no matter whether they had used it in their latest project). The respondents were people actively engaged in usability engineering and therefore the numbers in Table 1 are not representative of all development projects. On the contrary, most development projects do not currently have usability engineers on the development team, and prior research[1] has shown extraordinarily low use of usability methods in average development projects.

The engineers evaluated the methods' impact according to the following 1 to 5 scale:

Table 1. List of usability engineering methods showing the extent to which development projects actually used each method as well as the average rated importance of each method on a 1 to 5 scale: 1 indicates no impact and 5 indicates absolutely essential. The list of methods is ordered according to the approximate placement of the methods in the usability life cycle.

Activity or Method	Used on Project (percent yes)	Impact on Usability in General (average)
Visit to customer location before start of design	92	4.3
Task analysis of user's current task	69	4.7
Functional analysis of reason for user's task	46	3.8
Projection of evolution in user needs and abilities	54	3.4
Competitive analysis: Looking at existing competing products	77	2.9
Competitive analysis: Comparative analysis of competing products	23	2.8
Competitive analysis: User testing of competing products	23	3.1
Goal setting: Explicit priorities between usability parameters	38	3.3
Goal setting: Measurable levels specified for important goals	38	3.5
Participatory design: Real users involved during the design process	85	4.4
Coordination of the "traditional" user interface (screens, messages, and so on)	69	4.2
Coordination of the "total" user interface (manuals, training, and so on)	38	4.1
Use of a published vendor (or other de facto) interface standard	23	2.4
Use of in-house user interface standard or house style	69	3.5
Specification of a product identity	38	3.1
Use of large, general guidelines book	38	2.6
Use of category-specific guidelines for the type of product being developed	31	3.1
Listing and use of product-specific guidelines for the individual product	54	3.3
Heuristic evaluation (informal judgment based on guidelines)	46	3.3
Prototyping: Construction of paper mock-ups	46	3.0
Prototyping using computer tools	85	3.9
Empirical testing with real users as subjects	69	4.5
Measurements taken during test and compared with goals	31	3.3
Thinking-aloud experiments	31	3.1
Constructive interaction (two users work together)	38	3.3
Videotaping of user testing (as opposed to just taking notes)	23	2.9
Questionnaires to assess user attitudes toward the system	38	2.7
Iterative design	85	4.7
Explicit documentation of the rationale for the user interface design	46	3.3
Feedback from field use: Record user calls to hot line and so on	54	3.3
Method exists for users to provide direct feedback to developers	54	3.8
Field study/visit to customers to find out how the system is actually used	46	4.3
Logging of user actions on the system	38	3.0

(1) *No impact* on usability; it does not matter whether this is done or not.
(2) *Small impact*, but of no real importance.
(3) *Medium (and real) impact* on improving usability.
(4) *Major impact* on usability; should be done in most cases.
(5) *Absolutely essential* for improving usability; should be done in all cases.

The reason for surveying usability engineers instead of regular developers was to collect views founded on actual experience with the methods. Even if we cannot include a full-fledged usability engineering methodology in our development life cycle, we should at least choose the methods that usability engineers use or recommended the most. Actually, almost all the methods got an impact rating of at least 3, indicating that they were all judged as having a real impact on usability. The best approach would be to use as many methods as possible. But even if we have only limited resources to invest in usability, we should still consider some of the most important methods.

The top five methods according to frequency of actual use by the usability engineers are

(1) Visit to customer location before start of design.
(2-4) Iterative design, participatory design, and prototyping using computer tools.

(5) Competitive analysis: Looking at existing competing products.

The top six methods according to rated impact on usability are

(1-2) Iterative design and task analysis of the user's current task.
(3) Empirical testing with real users as subjects.
(4) Participatory design.
(5-6) Visit to customer location before start of design and field study/visit to customers to find out how the system is actually used after installation.

There is considerable overlap between the two lists, and there is in general a fairly high correlation between the use of the methods and their rated impact ($R = 0.71$). The three methods having the largest residuals in the regression analysis (indicating a mismatch between the scores on the two scales) are

• *Competitive analysis: Looking at existing competing products.* This seems to be done too much compared with the fairly low rated impact. From the impact rating, the regression "predicted" only 33 percent use.
• *Coordination of the "total" user interface* (not just screens but also manuals, training, and so on). This is done much too little compared with the high rated impact. From the impact rating, the regression "predicted" 64 percent use.
• *Field study/visit to customers to find out how the system is actually used.* This is done too little compared with the high rated impact. From the impact rating, the regression "predicted" 69 percent use.

The usability engineering life cycle requires the use of a large large number of usability methods to follow the complete set of recommendations given here. Often budget or time constraints will not allow the use of all the methods, but I recommend the following as a minimum:

- visit customer locations before the start of the project,
- do iterative and participatory design, and
- use prototyping and empirical tests with real users.

The single most important advice regarding usability engineering is simply to get started now. It is easy to get so overwhelmed by the complete usability engineering life cycle that we decide to wait until the next project, in which there will surely be more time for usability considerations. Unfortunately, the "next time" can easily be put off again and again. Instead, we should start with a few of the simpler methods right away and then gradually extend the scope of usability activities until the entire life cycle is covered. Such a gradual approach is more likely to succeed than an "all-or-nothing" approach.

The world is full of useless and frustrating software with functionality and user interfaces that could have been improved if their designers had used current usability engineering methods. The world is even full of frustrating alarm clocks, video recorders, and other appliances that could stand a dose of usability engineering. Please don't let *your* users suffer needlessly. ■

Acknowledgments

I thank Rita Bush, Tom Dayton, Tom Landauer, Alan McConkie, Amy Todres, Daniel Wildman, and several anonymous *Computer* referees for insightful comments on earlier versions of the manuscript.

References

1. J.D. Gould and C.H. Lewis, "Designing for Usability: Key Principles and What Designers Think," *Comm. ACM*, Vol. 28, No. 3, Mar. 1985, pp. 300-311.

2. J. Whiteside, J. Bennett, and K. Holtzblatt, "Usability Engineering: Our Experience and Evolution," in *Handbook of Human-Computer Interaction*, M. Helander, ed., Elsevier Science Publishers, Amsterdam, 1988, pp. 791-817.

3. J. Nielsen, *Usability Engineering*, Academic Press, San Diego, Calif., 1992.

4. B.W. Boehm, "A Spiral Model of Software Development and Enhancement," *Computer*, Vol. 21, No. 5, May 1988, pp. 61-72.

5. J. Grudin, S.F. Ehrlich, and R. Shriner, "Positioning Human Factors in the User Interface Development Chain," *Proc. ACM CHI+GI 87, Conf. Human Factors in Computing Systems and Graphics Interface*, ACM, New York, 1987, pp. 125-131.

6. J. Nielsen, ed., *Coordinating User Interfaces for Consistency*, Academic Press, San Diego, Calif., 1989.

7. J. Nielsen and R. Molich, "Heuristic Evaluation of User Interfaces," *Proc. ACM CHI 90, Conf. Human Factors in Computing Systems*, ACM, New York, 1990, pp. 249-256.

8. F.P. Brooks, *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley, Reading, Mass., 1975.

9. R.A. Virzi, "What Can You Learn from a Low-Fidelity Prototype?" *Proc. Human Factors Soc. 33rd Ann. Meeting*, Human Factors Society, Santa Monica, Calif., 1989, pp. 224-228.

10. J. Conklin and M.L. Begeman, "gIBIS: A Hypertext Tool for Exploratory Policy Discussion," *ACM Trans. Office Information Systems*, Vol. 6, No. 4, Oct. 1988, pp. 303-331.

11. M.M. Mantei and T.J. Teorey, "Cost-Benefit Analysis for Incorporating Human Factors in the Software Life Cycle," *Comm. ACM*, Vol. 31, No. 4, Apr. 1988, pp. 428-439.

12. C-M. Karat, "Cost-Benefit Analysis of Iterative Usability Testing," *Proc. IFIP Interact 90, Third Int'l Conf. Human-Computer Interaction*, IFIP, Geneva, 1990, pp. 351-356.

Jakob Nielsen is a member of the technical staff at Bellcore (Bell Communications Research). His research interests include usability engineering, hypertext, and next-generation interaction paradigms. His previous affiliations include the IBM User Interface Institute at the T.J. Watson Research Center and the Technical University of Denmark. Nielsen holds a PhD in computer science/user interface design and is a member of ACM, the Human Factors Society, and the IEEE.

Readers can contact Nielsen at Bellcore, MRE-2P370, 445 South Street, Morristown, NJ 07962-1910, e-mail nielsen@bellcore.com.