

Designing a USABLE Application: User Interfaces and Usability Testing

Jason Thomas, SAS Institute Inc., Cary, NC

ABSTRACT

The increased availability of graphical user interface (GUI) tools for application development calls for techniques to ensure that the applications are USABLE, and not just pretty or fancy. GUI tools, such as those available in SAS® application development using SAS/FSP® and SAS/AF® software with FRAME, have been created to develop better user interfaces for applications. This paper presents techniques and principles to aid the application developer in designing a usable interface.

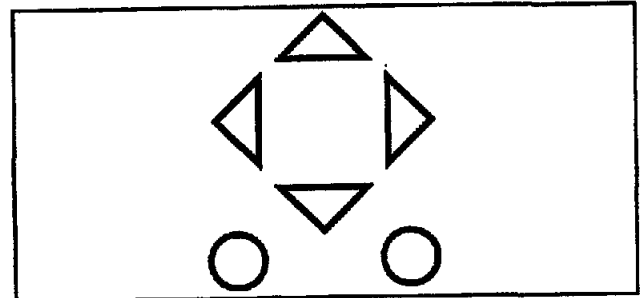


Figure 1. Television Remote Control

What is an Interface?

"To the users, the interface is the system" [Hix & Hartson 1993]. An interface is a means by which a person interacts with something to perform a desired function. We encounter such interfaces repeatedly in our day-to-day lives. For example, your telephone, your television remote control, your car stereo controls, your keyboard, and the buttons in an elevator are all interfaces that you encounter daily. Interfaces are evolutionary beasts, where survival of the fittest reigns. Remember the rotary telephone? What was once commonplace is now almost outdated due to a faster, easier-to-use interface: the push button telephone. With this in mind, it seems that manufacturers would be quite concerned with the interfaces of their products. Unfortunately, this is not always the case. After all, how many of us have trouble programming our own VCRs? Here are some recently encountered examples of some poorly designed interfaces:

- A remote control for a popular brand of television has just six buttons: one for power, one for a display to show time and channel, and four buttons in a north, south, east, and west configuration (see Figure 1). These four were likely for volume up/down or channel up/down, but which was which? I changed channels several times instead of changing volume levels! After inspecting the remote control, I found that the north and south buttons were for channel up and down respectively, while the west and east buttons were used to decrease and increase the volume levels. What does 'volume left (or west)' mean? Is that a balance control for a stereo television? Is this a good design?
- A typical conventional oven that consists of one oven on top of the other, with control dials *side by side*! Why? Does the left dial control the bottom or the top oven?
- A particular product in which the return key and the enter key behave quite differently, as described by Donald Norman in his book *The Design of Everyday Things* (1988). If the wrong key is pressed, the last few minutes of work is lost. When this problem was mentioned to the designer, he said that the company's secretaries had been using the system for some time with no complaints. When the secretaries were confronted, they admitted that they frequently lost work, but because it was their error, they blamed themselves! Who is really at fault? Is it really user error, or a poor design?

Why Are Products Not Usable?

Can you think of some poor designs that you have encountered? Why are these designs not usable? Jeffrey Rubin (1994) claims the following five reasons for hard-to-use products and systems:

1. The emphasis and focus while developing systems has been on the system and its functionality, not on the actual users of the system. He explains that "there has been an underlying assumption that since humans are so inherently flexible and adaptable, it is easier to let them adapt themselves to the machine, rather than vice versa." In the past, systems were purchased for their functionality, not their usability. A usable system makes sure that users can find and use those functions that meet their needs. The following example from Dumas and Redish (1993) illustrates this point well. In a discussion at a business meeting, people were complaining that their word-processing software did not do envelopes, but in reality, their word-processing software did indeed have that functionality. The users either did not know it, or thought it would be too difficult to learn. In short, "Building functionality into a product doesn't guarantee that people will be able to use it."
2. The typical user base has changed. In the past, designers were developing products for end users much like themselves. Today, this trend is far from the norm. Current users might range from scientists with extensive experience in their subject, to computer operators with years of computer experience, to an artist who may be a first-time user.
3. Organizations have treated user-interface development as common sense. "Usability is an elusively maddening concept" [Rubin 1994] about which everyone has an opinion. It is obvious that there is no one perfect interface for a given system. If you were to ask several users what was the most important characteristic in a usable interface, their answers would all be different.
4. Organizations employ separate teams for system development and fail to integrate them together. Often, separate teams will design separate components; teamwork is quite limited. Lack of teamwork often leads to an inconsistent feel, instead of one integrated product. It is essential for the individual components of a system to work well together for usability.

5. Developers are NOT users! The computer-knowledge level of a developer is much different than that of typical user. Take, for example, the Institute's vertical product SAS/PH-Clinical® Software. The application developers of this product do not completely have the knowledge and skills of the physicians, clinical data analysts, and statisticians that use the software every day. It is crucial that developers involve *real* users in the design of the product if they are to meet their needs. The developers of SAS/PH-Clinical were concerned about what real users did, so they recruited people from the pharmaceutical industry with varying experience levels and performed usability tests with these people. Each developer of SAS/PH-Clinical got to actually see and hear a typical user perform several real life tasks on the product. According to Linda Lunney, a developer of the product, "valuable information about the interface and the software was obtained." This type of study helps bridge the tremendous gap between users and developers, and thus helps SAS/PH-Clinical become a more usable product. The employers of the usability subjects actually volunteered their employees for the test. Users want the product to be usable, and they love to give input!

In summary, there are many reasons why many products are not usable. Let's look more closely at what 'usable' means.

What Is a Usable System?

Usable means "that the people who use the product can do so quickly and easily to accomplish their own tasks" [Dumas & Rubin 1993]. Usable is a subjective term, and a usable system is more than 'user friendly.' Usability is the ability to function as an efficient tool, enabling users to perform their tasks. Jakob Nielsen (1993) breaks usability into five distinct attributes: learnability, efficiency, memorability, errors, and satisfaction:

1. Learnability

Learnability of a system describes how easily new users can rapidly start completing their tasks. A highly learnable system allows users to become rapidly productive. All of us now expect rapid learning, whether learning a new tracking system, an email utility, or a VCR. Likewise, many users are becoming accustomed to not having to pick up a manual to figure out how to begin. Learnability is also fairly easy to measure. By presenting novice users with a task and observing and measuring how long it takes them to complete the task, you can quickly and easily determine the learnability of a system.

2. Efficiency

A system needs to be easy to learn and efficient to use. Once users have learned the system, a high rate of productivity can be achieved. However, a trade-off point often occurs between learnability and efficiency (see Figure 2). Users of a highly learnable interface will become somewhat productive quickly, but they soon will reach a point where their productivity levels off. The user of a harder-to-learn, yet highly efficient interface will not initially be as productive, but eventually they will become more productive. A good method to ensure that both the novice and the expert user perform at maximum efficiency is to have different 'user level settings.' An expert user might not need certain

prompts, reminders, and messages that are essential for the novice users. By offering the ability to change user level settings, we accommodate both the expert and novice users. Another popular efficiency method is to include accelerators in the user interface. Accelerators allow a user to perform tasks quickly and easily. Typical accelerators include:

- Function keys
- Activating an object by double clicking on it
- Command abbreviations, for example, FSE for the FSEDIT command
- Mnemonics on pull-down menus that provide users with a short cut for selecting a function (available via PMENUS on some hosts)
- Toolbars which make popular functions easily accessible.

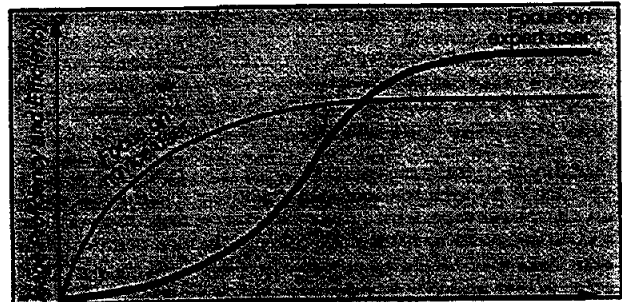


Figure 2 Learning Curve of a Hypothetical System. (Reprinted with permission from Jakob Nielsen, Copyright 1993 by Academic Press, Inc.)

3. Memorability

A system has high memorability if the casual user returns to the system after some period of non-use and does not have to relearn the system. Casual users only use systems periodically. These users have used the system before, so they do not need to relearn the system from scratch. A marketing analyst who runs quarterly sales reports is an example of a casual user.

4. Errors

A usable system should have a low error rate. In all systems, user errors are unavoidable, but through good interface design, user error frequency can be drastically reduced. You can measure error rates by simply counting the number of errors or undesirable actions made by users while performing a task. When errors do occur, it is essential to provide a means by which the user can recover. If you have been caught in an infinite loop, or have been unexpectedly 'kicked out' of a system without notice or explanation, you may remember the resulting frustration that occurred. Potentially dangerous or unrecoverable actions such as record deletion should give users an opportunity to confirm and provide a way to cancel.

Good error messages are also crucial for usability. Four basic rules apply to constructing error messages [Shneiderman 1982]:

- Error messages should be phrased clearly. Avoid vague messages such as "Error #3304."
- Error messages should be as precise as possible. For example, do not use "Employee not found," but "Employee Jason Thomas was not found in the employee data."

- Error messages should constructively attempt to help the user solve the problem. For example, if the input was not valid, the system could provide a list of valid values to choose from.
- Good error messages do not blame the user or sound intimidating. Messages such as "USER ERROR," or "UNKNOWN COMMAND" are accusatory. Users do not make errors intentionally, so they do not wish to be reprimanded.

5. Satisfaction

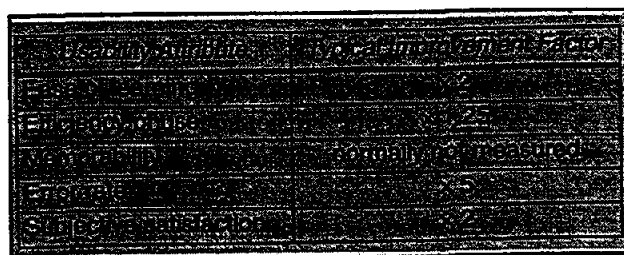
A system that is pleasant to use has many benefits. Not only do the users enjoy it, but their overall perception of the software quality is enhanced. Satisfaction is a not just of primary importance in the success of video games and entertainment software, it plays an important role in business-related software as well. Marketing professionals can confirm how important 'word-of-mouth' is among customers.

As you can see, there is more to a usable system than just being 'user friendly'. A usable product is one that is easy to learn, efficient, memorable, has a low error rate, and is enjoyable. Each of these factors plays an important role in a product's usability and success.

Some usability attributes may be more important to the user than others. For example, a library resource-locator application should quickly allow users to find desired material. For such a system, learnability may be the most important usability issue to target because the user's goal is to quickly locate a resource without having to access help screens or look in a manual. The most important concept is that system design must be centered around the user.

What Are the Gains and Costs of Usability Engineering?

Usability Engineering is a combination of user-centered design and usability testing. The gains received from usability engineering are quite impressive (see Figure 3).



Measurable Usability Attributes	Normally Measured	With Usability Engineering
Learnability	100%	200%
Efficiency	100%	125%
Error rates	100%	50%
User satisfaction	100%	200%

Figure 3. Measurable Usability Attributes. (Reprinted with permission from Jakob Nielsen, Copyright 1993 by Academic Press, Inc.)

Figure 3 shows that usability engineering can double learnability, give a 25% increase in efficiency, reduce error rates by 500%, and double the user's subjective satisfaction with the product. These are quite outstanding improvements!

The advantages of a usable product are many. In an internal (in-house, not-for-profit) application, a reduced error rate, increased efficiency, and smaller learning curve greatly increase not only the productivity of the user, but also reduce the support and training costs associated with supporting the system. If an internal user has to call a support person, the company is paying BOTH people for their time! An easily learnable product is definitely cost effective in an environment with a high employee turnover rate.

In an external (sold for a profit) application, there are many different gains from having a highly usable product. Of course, the desired bottom line is an increase in sales. "Usability is typically worth about 30% of a product's review weight" [Nielsen 1993]. Many MIS executives ranked usability to be a more important factor for a product than functionality, performance, or support [Flanagan 1994]. Increases in subjective satisfaction help sell not only current products, but future products as well. Another big savings for external, as well as internal applications, is the reduction of expensive support costs. Other savings gained from producing usable products [Dumas & Redish 1993] include:

- Enhanced company reputation
- Reduced need for updates and maintenance releases
- Reduced training costs
- Easier creation of documentation.

Great financial gains from usability engineering have been documented. In an IBM Cost/Benefit Study, the cost of usability engineering for an internal product was \$20,700. However, the resulting increased usability saved each user 4.67 minutes daily, which resulted in savings of \$41,700 the very first day the product was used! Similarly, a large product (over 200,000 users) cost \$68,000 for usability engineering; the benefit during the first year of implementation was \$6.8 million [Karat 1990], a benefit that was 100 times the cost! Jakob Nielsen mentions that most companies get back at least "50-100 times what they paid" (for usability engineering).

There are also many gains from usable systems that are not necessarily related to financial costs. For example, a pilot in an emergency situation does not have time to look in a manual for seldom-used functions. Usable pilot controls are crucial. Usable systems are also essential for a surgeon who uses multi-million dollar equipment in life-saving surgery, military personnel who use billion dollar equipment defending our country, and even the person in a library trying to find a book. No direct financial gain or cost is assumed, yet usability is of extreme importance in these examples.

Why Is Usability Engineering Not Performed?

With the knowledge that 30% of a product's review weight is based on usability, do all companies strive to make their products as usable as possible? No. Some of the reasons companies are *not* practicing usability engineering [Flanagan 1994] include:

- Deadline pressures
- Usability addressed too late in the development lifecycle

- Functionality or technology drives decision-making in development rather than usability
- Management does not support usability
- Usability considered an unknown, inexact concept
- Lack of knowledge about usability
- Aesthetics are more important than usability.

Usability Engineering: User Centered Design and Usability Testing

At the core of usability engineering is a concept called *user centered design* (UCD), which puts users at the *center* of the design. User centered design fits nicely in the software development lifecycle and puts emphasis on making the design fit the user, not the user fit the design. This principle involves the users from the start, and iteratively involves them throughout the process. UCD molds the product BEFORE changes get costly. "Usability is not a surface gloss that can be applied at the last minute" [Dumas & Redish 1993]. Usability has to be built in the beginning of the development lifecycle and practiced throughout the development of the product.

User centered design is comprised of the following steps:

1. Knowing the User
2. Task Analysis
3. Goal Setting
4. Prototyping
5. Usability Testing
6. Retesting and Iterative Design

1. Know the User

The importance of knowing the user cannot be overstated. As previously stated, developers are NOT users. Users may include current users of the system, users of a similar system, or users who will be using the product in the future. User experience levels vary greatly and may include general computer, application, and real-world experience. Effective user centered design does more than classify or identify the user; it *understands* the user. Knowledge of the following information is of great importance in understanding the users of the system:

- Employee turnover rate
- Available training
- Frequency-of-use of the product
- What other systems are used.

Another important factor to consider is knowing what type of pressures a user might face. Time may be the largest pressure for a newspaper publisher, while accuracy might be an important issue for a financial consultant. The environment a user works in also needs to be known. For example, a library system should not beep when incorrect input is given.

The most informative way to get to know users is to actually witness them in their real work environment doing their day to day jobs. Other techniques for getting to know the users include interviews, market analyses, and questionnaires. Unfortunately, it is often difficult for developers to gain actual contact with users; site visits may be costly, or marketing staff may want to limit developer contact with customers. Any contact is better

than none, but "new insights are almost always achieved by observing and talking to actual users in their own working environment" [Nielsen 1993].

2. Task Analysis

Task analysis is performed so that developers can understand the nature of the work that users need to do with the system - what their job is. "Many user interaction developers claim that insufficient or inaccurate task analysis is responsible for many terrible designs" [Hix & Hartson 1993]. Important issues to discovered in a task analysis are: What are the goals for the system; How are users currently accomplishing their tasks; and What do users need to accomplish their tasks.

• Goals of the System

Focus on what the user wants to accomplish with the system. Users want the system to allow them to do their job easily and efficiently, but business goals may also come into play. The system may eventually need to integrate with another product, or adhere to some set of guidelines. Product goals are not necessarily made by the user, often management sets these goals as a part of the big picture.

• Current Task Flow

Perhaps most importantly, task analysis must examine what users currently do to accomplish their tasks. This examination involves producing some kind of step-by-step task list in which each step is a separate, detailed sub-task. Once all of the sub-tasks have been produced, make sure that all of the interdependencies of these sub-tasks have been identified to ensure correct task flow. Also, note any exceptions from the flow that are known to occur. It is much easier to plan ahead for exceptions than to have to change the design later. Find out if the users have workarounds that they use when exceptions to work flow occur.

• User Needs

Determine what users *need* to accomplish their task. Often users can tell you what they need but currently do not have, however keep in mind that users may not be aware of what functionality can be made available to them. Also, review other existing products to get design ideas. After task analysis is complete, always validate your analysis to ensure correctness.

Effective task analysis includes real user involvement, not managers or coworkers (unless they are real users). A good user sample, with a mixture of experience and expertise, greatly increases the effectiveness of the task analysis. Often, there is a 'lead user' who has much experience with the current application or the current process. This type of user is invaluable. Novice users also play an important role, because all users will be novice users in the beginning. The important concept is to get a good mixture of users, for "No one individual can represent all users" [Dumas & Redish 1993].

3. Goal Setting

Goals must be set in order to determine if the system design is successful. Setting a usability goal of 'user-friendliness' is not a sufficient goal. Usability goals are best when they are specific and measurable, for example: "Users will be able to check out a book online in less than three minutes," or "Users will average less than 5 errors per hour." Once these goals are set, you

should use them to drive the development decisions for the product. When setting usability goals and throughout the design process, consider who the users are and what their tasks are. Recall that some characteristics of usability are more important to a product's success than others. For example, if the product is an installation utility, it most likely will be used quite infrequently - thus learnability is vital to its usability. If the product is an online data entry form, a low error rate might be important to save staff time in making necessary corrections. If you have specific and measurable quantitative usability goals, you have a clear basis for evaluating if the goals are met [Dumas & Redish 1993].

4. Prototyping

Prototyping is used to explore different design possibilities and to test the possible designs with the users. Prototyping is a method of quickly and inexpensively designing rough drafts of a system that can be easily changed over and over again. A prototype should be designed with the fact that it WILL get changed and possibly even totally thrown away. It is both time and cost effective to make changes at the prototyping stage in the software development cycle (see Figure 4).

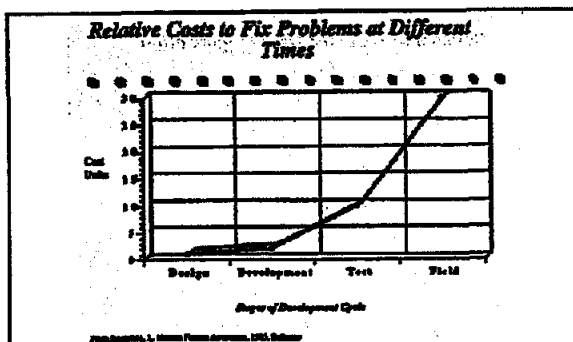


Figure 4. Relative Costs to Fix Problems at Different Times. (Reprinted with permission from Bellcore, Copyright 1994).

There are many different techniques for creating prototypes, but two main groups: low-tech prototypes and high-tech prototypes. *Low-tech prototyping* does not involve a computer, while *high-tech prototyping* does. One low-tech prototyping method is the paper and pencil method, which does not require in-depth training, is very inexpensive, and allows rapid changing of the prototype. Because computers are not involved, everyone is on the same level; those who are not experienced computer users do not feel intimidated. Paper-and-pencil low-tech prototyping requires users to create paper mock-up screens of the desired system. Materials to create the mock-ups consist simply of sticky pads, tape, rulers, colored markers, colored papers, and other supplies as needed. The concept is to create the screens from these supplies. If a push button needs to be moved to a different location, simply move it on your paper mock-up.

When low-tech prototyping, form different groups of prospective users who create separate prototypes, then combine the best of the prototype results from these groups. For added realism during low-tech prototyping, use computerized drawings, screen captures, or photocopies of the different user interface widgets (buttons, listboxes, etc.).

High-tech prototyping uses computer interface design tools to develop the prototypes. Because SAS/AF and FRAME technology is well suited for rapid application development, it serves as an excellent high-tech prototyping tool. Although developing high-tech prototypes require more time, skill, and effort than low-tech prototypes, they give the users a more realistic feel of the system. Much of the output from high-tech prototyping can be used in the real design.

Do NOT attempt to design the interface while creating the high-tech prototype. A prototype is temporary, and it WILL be changed or thrown out! Do not waste costly time on details that will be handled later when the design is more concrete. Some shortcuts to save time and resources include:

- Using dummy data
- Having the system indicate what action it would take by displaying a message rather than actually coding the action
- Not worrying about efficiency
- Not programming for exceptions (such as leap years) [Nielsen 1993]
- Not worrying about detailed aesthetics (colors, spacing, and the like).

Concerns about details will be addressed later, but not until the design has been more firmly defined.

High-tech prototyping, while more realistic and somewhat reusable, does have some drawbacks. It is more time consuming, response times of the real system cannot truly be simulated [Dumas & Redish 1993], and changes are not nearly as easy to make as compared to low-tech prototyping. First, do several designs with low-tech prototyping, evaluate those designs, then repeat the procedure using high-tech prototyping, each time using different REAL users. During your prototyping, several iterations can provide many different user perspectives and possibly several different designs.

5. Usability Testing

After high-tech prototype development is completed, developers have a tested design in which to develop the application. However, we are not finished! The final stages for usability engineering are no less important than the initial ones. Usability testing is an extremely important piece of the software development lifecycle, and the benefits have been recognized (Figure 2). Usability testing is used to evaluate a system to see if it meets the previously defined usability goals, as well as to help confirm that the system and design are good. Usability testing may also show problems so that management can allocate the necessary resources to fix the problems. However, it is important to realize that each individual iteration of testing, evaluating, and designing may not always solve a usability problem. Individual iterations and testing may even introduce new problems [Nielsen 1993]. It is quite common that an attempt to enhance one component of the interface may have an adverse effect on another. For example, enhanced data validation, while reducing errors, may slow data entry performance.

When usability testing during the iterative design process, it is not necessary to always involve real users. This may not be feasible due to time or financial constraints. As an alternate solution, users can still be involved via interviews or electronic questionnaires when design decision input is needed. However each major iteration should be tested with real users.

How do we usability test? Usability testing has been covered extensively in many books, and it is beyond the scope of this paper to explain the process in great detail. Jeffrey Rubin's *Handbook of Usability Testing* (1994) contains some excellent material for orchestrating a usability test. In short, the usability testing process falls into three parts: performing the test, evaluating the results, and using the results to direct the next iteration.

Performing the Test

The usability test is simply an observation of real users performing real tasks on the application. A closer look at the individual parts of a usability test will help illustrate how it is performed.

- **Who:** The users being tested must be real users with a mix of expertise and backgrounds. In determining the number of users, three to seven users provide the best cost-to-benefit ratio [Nielsen 1993] (see Figure 5).
- **What:** Tasks that users are asked to attempt should mimic those that they do in their day-to-day jobs as outlined in the task analysis.
- **Where:** Many companies have designated labs for the sole purpose of performing usability tests. An ideal usability lab includes sophisticated equipment such as:
 - one way mirrors in which developers can witness the user without the user feeling 'watched'
 - a video camera connected to or facing the screen to record the user's actions (errors, steps to complete a task) within the application
 - a video camera set up to record the desktop (mouse actions, documentation uses, etc.)
 - a video camera to record the user's expressions (voice, facial expression, gestures).

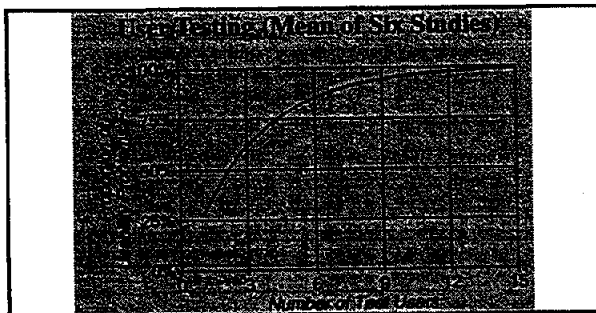


Figure 5. Proportion of Usability Problems Found Per User. (Reprinted with permission from Jakob Nielsen, Copyright 1993 by Academic Press, Inc.).

However, most usability facilities are not equipped with such elaborate equipment, and such equipment is not needed to perform a successful usability test. Any room will suffice for usability testing, although users should be made to feel as if

they are on the job. However, video taping is highly recommended so that users can be observed after the test and problems can be more closely analyzed. Often developers do not realize how misleading their interfaces are until they actually SEE someone having difficulty.

- **How:** A process called thinking aloud "may be the single most valuable usability engineering method" [Nielsen 1993]. Thinking aloud is simply a method where users verbalize their thoughts as they make decisions while performing tasks. This method helps the developers understand how the user interprets interface components, as well as outline the logical flow as the users see it. Thinking aloud is very difficult for some users; frequent reminders to verbalize are often necessary. It is imperative that the test monitor does NOT interfere with the user, especially if the user is struggling. When users struggle, they provide just the kind of information that developers need!

Evaluating the Results

The usability test yields vital information about the usability of a system, including:

- Error rates
- Time measures
- Help system evaluations (how it worked)
- Most frequent paths taken by the users
- Signs of user frustration
- Unfinished tasks.

After the test, it is also helpful to have users fill out a post-test survey, asking questions on specific pieces of the interface, and even asking for recommendations for solutions to pieces they found problematic.

Using the Results

Data is not just collected for fun! Simple data analysis and comparisons will indicate a product's improvement or regression. Remember those specific and measurable usability goals created in the beginning of the design process? Now is the time to compare those goals with the results of the usability tests to determine if the product meets its desired usability level. These results are what should drive the design through the next iteration. Some components of the interface may have suffered due to changes in another; some changes may have failed miserably. At this point, the usability problems should be evaluated and prioritized based on their severity.

6. Retesting and Iterative Designing

Once usability results are evaluated, problem areas of the system should be redesigned, retested, and finally evaluated again via the iterative design process. While the gains of iterative design are quite tremendous, you should keep in perspective that you will never have enough time, money, and user access to do everything you want to do. The most important thing is to do *something*! The product WILL be tested for usability eventually, even if you do not do it yourself. Users will do the testing for you.

CONCLUSION

The concept of usability is both quite complex and somewhat subjective. At the heart of this concept are the users, who simply have a task to do and a need to complete the task easily and efficiently. By getting to know the users and what they need to do, and by involving them early and throughout the iterative design process, a more usable product can be obtained. Remember, develop systems for the USER!

In closing, recent software development has devoted an average of 48% of the code to the user interface [Myers & Rosson 1992]. Combining this with the fact that usability is worth about 30% of a product's review weight, it is amazing that only "6-10% of development budgets are allocated to usability" [Nielsen 1993]. SAS/FSP and SAS/AF with FRAME technology give us a whole world of graphical user interface tools to help make our applications more usable. It only makes sense to work hard at ensuring usability. In fact, we must ask ourselves, "Can we afford not to?"

REFERENCES

- Bellicore, *Introduction to User Centered Design* Course Notes, 1994.
- Dumas, Joseph S. and Redish, Janice C., (1993), *A Practical Guide to Usability Testing*, Ablex Publishing Corporation.
- Flanagan, George, IBM Consulting Group, 'Quantifying the Positive Effects of Usability', Presented at *Triangle Chapter of ACM/SIGCHI*, October 25, 1994.
- Hix, Deborah and Hartson, H. Rex, (1993), *Developing User Interfaces Ensuring Usability Through Product & Process*, John Wiley & Sons, Inc.
- Karat, C. Cost-benefit analysis of iterative usability testing. *Proc. IFIP INTERACT'90 Third Intl. Conf. Human-Computer Interaction*, 351-356.
- Mehlenbacher, Brad, (1993), "Software Usability: Choosing Appropriate Methods for Evaluating Online Systems and Documentation," *Conference Proceedings of SIGDOC '93*, 209-222.
- Myers, B.A., and Rosson, M.B. (1992), Survey on user interface programming. *Proc. ACM CHI'92 Conf.*, 195-202.
- Nielsen, Jakob, (1992), "The Usability Engineering Life Cycle," *Computer*, 12-22.
- Nielsen, Jakob, (1993), *Usability Engineering*, Academic Press, Inc.
- Nielsen, Jakob, (1993), "Usability Engineering," *User Interface Strategies '94*, p35-54.
- Norman, Donald A., (1988), *The Design of Everyday Things*, Doubleday Currency.
- Rubin, Jeffrey, (1994), *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests*, John Wiley & Sons, Inc.
- Shneiderman, B. (1982). Designing computer system messages. *Communications of the ACM* 25, 9 (September), 610-611.
- For Further Information Contact:*
Jason Thomas, E315
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
sasjzt@unx.sas.com
- SAS, SAS/AF, SAS/FSP, and SAS/PH-Clinical are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. IBM is a registered trademark of International Business Machines Corporation. * indicates USA registration.