User Interface Design Explained

Douglyss Giuliana

Introduction 3

Fundamentals of User Interface Design 3

Benefits of Usability 3
Minimize User Interface "Bugs" *4*
Save Time and Money *4*
Increase User Productivity *4*
Reduce Training and Support Needs *4*
Improve Product Quality *5*
Improve User Satisfaction *5*
Market It *5*
A User-Centric Design Approach 5
Human Factors Goals *5*
User Interface Design Principles 6
Consistency *6*
Redundancy *6*
Forgiveness *7*
Feedback *7*
Simplicity *7*
Interaction *7*
Directness *8*

User Interface Design Methodology 9

User Identification 9
Task Analysis 9
User Interface Design Guide 10
User Interface Design 11
Iterative Prototyping 11
Usability Testing 12

What's Next 13

Introduction

The purpose of this document is to provide developers and analysts with the background, methodology, and frame of reference to create an effective user interface design, regardless of the hurdles. Much of the information presented here will cover the "why," "what," and "how" of interface design, the general knowledge that applies to all projects. Using this manual as a reference, each project may then create a "User Interface Design Guide" which presents the specifics for that project. The design guide should include project-specific information based on the users, tasks, and tools, and should dictate the specific controls, sizes, icons, vocabulary, and interaction that is standard in the project's interface.

The topic of interface design was introduced by the Air Force in the mid-seventies, but has been hot since about 1990. Over the past few years, it has become one of the most important criteria for judging software. Nearly every software ad has the words "user-friendly" or "ease of use" plastered on it somewhere. Every trade rag software review tests product usability. It is estimated that 15% of a magazine software review is based on the usability of the interface (Nielsen, 1993). **Marketing is pushing it, customers are demanding it, now developers must provide it.**

User interface design is not a difficult subject; it is, however, a complex subject. With the variety of situations developers are put in, the incredible number of designs possible with today's tools, and the ever-changing computer industry, designers must be ready for anything. There are a myriad of books on this subject that try to tell developers how to design a great interface. Rarely do developers have the opportunity to create such a great interface. Rarely is everything as straightforward as the books would describe. With the emphasis on time and money, the immediate desires of the client, and volume of project requirements, following "recipe" interface design is nearly impossible.

Many developers comment that good user interface design is just common sense; everyone can do it. Effective user interface design is straightforward, but some basic knowledge is required to give you the tools and mindset to design with usability in mind. You wouldn't send a boxer into a Karate match just because he said he knew how to kick; it requires some training. A paradigm shift is required to allow the developer think like a Human Factors Engineer.

Fundamentals of User Interface Design

Benefits of Usability

It is hard to sell usability in all but the largest of companies. Incorporating usability into a product is seen as expensive and providing no solid benefit or payback; "it just makes the product look nicer," they say. Those that do see some of the benefits of a user-friendly product don't believe that usability will adequately pay back the initial expense.

This fact has, of course, been researched and tested over and again. While usability does cost more in the initial phases of the project, costs level-off before release and the savings continue for the life of the product (Mayhew, 1994). The reasons for this are simple**: without attention to usability users are less productive, products take more time to develop, require more training and support, and are less attractive to customers.** So, why usability?

Minimize User Interface "Bugs"

A software bug is a piece of code that does not do what it was intended to do. Similarly, a user interface bug is a user interface design that does not do what it was intended to do -- to provide the user with a clear, intuitive interface that allows the user to be productive. Examples of such a bug would be poorly organized data, the use of ambiguous terminology, and not providing useful error messages. These UI bugs are as important as software bugs because they prohibit the program from being used with the efficiency and accuracy intended and often required. By incorporating usability into the design process, these bugs can be avoided before development takes place, thus producing a higher quality product with fewer costly changes.

Save Time and Money

Let's face it, in most projects the developers design their own screens. Developers are assigned a set of screens they are responsible for, they retreat to their cubes, and design and develop the screens, individually. As the project progresses, developers come together to combine their screens into the final product. They begin to see inconsistencies and duplications. They realize that the screens don't look like they "go together." They see that they used different vocabulary, different icons, and different looks. At this point, they can do two things. They can leave the product the way it is, sacrificing user satisfaction and future sales. Or, they can fix it, making changes to the already developed code. These are both costly options. Integrating usability into the process will identify these bugs during design phase, where changes are quick and inexpensive. It is estimated that changes early in the development cycle cost one-fourth (1/4) of what changes late in development would cost (Mantei and Teorey, 1988). A simple set of standards, identifying common elements, and an ongoing dialog would eliminate this costly mistake. Get your development right the first time, during design.

Increase User Productivity

In almost every business, productivity equals profitability. Companies want their employees to be as productive and efficient as possible, especially in these days of downsizing and rightsizing. In order for employees to be productive, their tools must help them be efficient and accurate. By creating intuitive, easy-to-use interfaces, developers create such a tool. If the user needs to constantly check the documentation or online help to figure out the interface, or is making errors that slow them down and require correction, then that tool is hurting the companies bottom line. If usability methods can improve a user's productivity by one second per screen, that will have a great impact when one considers the number of screens a user processes a day, the number of users, the number of days they work, and their compensation. **Usability equals productivity equals profitability.**

Reduce Training and Support Needs

Training and support costs can have a great impact on a companies bottom line. Imagine the training and support costs for a fifty person data entry department that is moving from the mainframe to Microsoft Windows©. Consider the training required for these workers, and the training for Windows products. Consider the support calls the Help Desk would have to handle. This is surely an incredible cost. Usability focuses on designing for the user. By creating an interface that builds on what the user already knows about Windows, related products, and their jobs, you can create an interface they better understand. The better they understand how to use the product to complete their jobs, the less money spent on training and support.

Improve Product Quality

To rate a product's quality, you must consider product functionality, efficiency, accuracy, cost, appearance, and user satisfaction. As you have already seen, usability can affect five of these. If usability is threaded throughout the design and even development process, product quality will inevitably be built in. Quality products can only come from a quality methodologies.

Improve User Satisfaction

User satisfaction is based on aesthetics, functionality, and success. The first impression a user gets about a product is from the look of the user interface. Sloppy products have sloppy interfaces -- mis-alignment, poor wording, confusing navigation. These can lead to a negative impression even if the functinality is strong. If a user can interact easily, and perform their job successfully, they will surely be pleased with the product. A user-friendly product can greatly impact this satisfaction.

Market It

Many companies have realized the importance that users put on usability these days. Car commercials use the words "ergonomically designed." Electronics companies tout "easy to use" gadgets. And software companies shout "user-friendly" software. The users want it, and expect it. If you can offer it, you will have an advantage over those that do not.

A User-Centric Design Approach

"A well-designed user interface is based on principles and a development process that centers on users and their tasks." (Microsoft Corporation, 1995) This quote is the first sentence in the first chapter of The Windows Interface Guidelines for Software Design. A user interface is just that -- an interface through which the user receives some stimuli and provides some response. The user is the target of the information and the driver of the system. This fact is often lost because interfaces are designed and developed by developers, not users. Developers have a different view of the product, a different skill set than the users, and often enforce their own desires rather than those of the end users. **Only the users know what they need and what they want; and the only way to find out what the users need and want is to ask the users.**

Putting the user at the center of the design approach greatly improves the chances of creating an intuitive, efficient, and effective interface. "Know thy users" has long been the cry of Human Factors Engineers. This parallels the "know thy opponent" idea in sports. Different opponents have different strengths, different skill sets, and different strategies; every game can not be approached the same way. Likewise, you can't apply the same design to every set of users. An understanding of the target users and their tasks must be the cornerstone of the design process.

Human Factors Goals

Human Factors Engineering is based on 3 goals:

1. Provide an interface that is intuitive to the users.
2. Provide the user with the easiest interaction possible.
3. Help the users complete their tasks.

These goals are straightforward. First, create an interface that the users can readily understand, with a minimum of training and a limited use of help and documentation. Not only will this save money on training and support costs, it will mean users are being productive from day one. Second, make it easy for the users to get their job done. Do not require the user to remember codes or data, and keep the number of actions required to complete a task to a minimum. Third, do not just make it easy for the users to complete their jobs, make it really easy. Help the users perform their jobs, do not just allow them to do it. Provide defaults where appropriate, guide the user from step to step, and do their job for them where you can.

This may seem like a tall order, but when the user is at the center of the design process, much of this will come for free.

User Interface Design Principles

Remembering the quote mentioned above, “A well-designed user interface is based on principles...” This section discusses the principles upon which design decisions should be made.

Consistency

Consistency is the most important principle in Human Factors Engineering. Consistency allows users to leverage what they already know when learning a new task or interface. When interfaces provide similar layout, terminology, interaction, and navigation, the user can spend less time learning each interface’s intricacies and more time being productive.

Consistency must be applied at several levels. First, you must maintain consistency within an application. Application-level consistency includes a similar look-and-feel for all windows, consistent use of metaphors, and navigation methods. Second, the application must be consistent with the system and other applications on the system. This includes a common look-and-feel, common functions such as saving and exiting, accelerator keys and mnemonics, and standard methods such as navigation, selection, and editing. Finally, window-level consistency must be maintained. Each window should have common control placement, alignment, and grouping, and should use standard terminology, which includes both words and icons.

Redundancy

Redundant cues can be found wherever you look. A stop sign has a specific shape, a specific color, and the word “Stop”. Using multiple cues increases the likelihood of recognition. If all traffic signs were the same shape and color, and simply had a different word on them, driving would certainly require much more concentration. The same holds true for user interfaces. Provide multiple cues wherever possible. For example, when an error occurs, pop up a dialog and make the “beep” sound. If the user isn’t looking directly at the screen, they will still notice the error.

Redundancy goes beyond cues and includes interaction methods as well. Every menu item and button should also be accessible by keyboard. Every mouse action should have a keyboard equivalent. This will allow people to use the method they are faster or more comfortable with. You wouldn’t want a data entry clerk constantly lifting their hands off of the keyboard because they needed to use the mouse to push the save button.

Accounting for cultural and personal differences, as well as limitations, is why redundancy is so important. It allows users to work in the way they are most comfortable and most productive.

Forgiveness

Regardless of how well an interface is designed, users are going to make mistakes. Whether pushing the wrong button or entering the wrong data, mistakes will occur and need to be handled. The best interfaces will help users avoid even making these mistakes. Enabling buttons only when appropriate and prompting before committing actions provide reminders to users about the effects of their choices. However, when mistakes do occur, it is important to be forgiving. Allow users to back out of or undo actions, especially those that are destructive.

Users like to explore a systems capabilities when they are learning it. They will click on each button to see what it does and play with each screen. Let them explore without risking the integrity of the data or the system by allowing them to undo their actions or cancel a series of choices. They will learn the system by trial-and-error, and will become comfortable interacting.

Feedback

Every action that the user performs should provide some feedback immediately, and at least within a few seconds. Whether the action is clicking a menu item, pushing a button, or selecting some text, provide a response to the users. Users rely on this feedback to let them know that the application received the input. If the user does not believe their action was accepted by the program, they may do it again, causing it to happen twice; they may think the application is dead, and restart; or they may wait, hoping a response comes eventually. Each of these possibilities makes the user less efficient and more uncomfortable with the application.

Feedback can be visual, audio, or both. If an action will take more than a few seconds to complete, provide the user with some idea of how long it may take, and keep them updated on the progress if possible. Feedback can even be provided that describes what will happen if the action is carried out. Also, double check destructive actions with the user before they are committed.

Simplicity

An easy way to make an interface intuitive and easy to use is to keep it simple. Clearly show the user what functionality is available, but avoid distracting the users with unnecessary information. Unfortunately, functionality and simplicity are inversely related -- the more functionality the less simple. These two factors should be carefully balanced in the design.

Show users the information they frequently need or need to access urgently. Give them the option of progressively seeing more information if needed. This will allow users to easily find and digest the information they need. Avoid the use of art or decoration if it does not contribute to the understanding of the information.

Interaction

Interaction with the system should allow the user to be an actor, rather than a reactor. The user should be in control of the system, driving the actions the system performs. Avoid

controlling the user's actions, requiring them to react to the system's actions, at the system's speed, and with the system's methods.

Allow the users to personalize the system. Users will have their own preferences, based on their skills and tastes. Where appropriate, let the users customize settings such as defaults, colors, fonts, and options.

Directness

The most intuitive user interfaces allow users to directly manipulate the data and objects. This direct manipulation mimics the way people interact with objects on a daily basis. Offering this level of interaction in a user interface creates an interface that is easily learned and understood.

Allow users to directly manipulate the objects on the interface. Provide interactions such as drag-and-drop. Imagine a drawing program with a command-line interface or a word processor that did not show the text using the formatting the user applied. Such an "indirect" interface can be quite confusing and ignores the needs of the users.

User Interface Design Methodology

Remember the importance and benefits of the user-centric design approach mentioned above. The first step, then, is to identify and quantify the users, and analyze the tasks they need to perform. Design standards can then be developed to ease the design process. With this information in hand, the iterative process of design, prototype, and test continues until a satisfactory interface is achieved.

User Identification

The first step is to identify the users. Who are the people who will interact with the system? In particular, the following questions should be answered:

1. How much and what computer experience do they have?
2. What other software programs will they be using?
3. What else do they do when not using the computer?
4. How do they perform their jobs now?
5. What do they have specific knowledge about?
6. What computer equipment do they have?
7. What is their screen resolution?

Each of these questions will be useful when creating a design guide and during the design phase. If they are expert computer users, they know more complex navigation techniques. The software design should be consistent with any other software they use. Familiar metaphors can be used to simplify the user interface. Knowing their computer equipment, metrics can be created to limit window size and application complexity. Without knowing this information about the users, developers may mistakenly use information about themselves.

This information can be gathered through user interviews, user questionaires, and user observation. User questionaires can be administered to a large number of users quickly. Questionnaires are best used for finding out user's hardware configurations, other software they use, and computer experience. User interviews can provide more detailed information, including how users currently perform their jobs and what other knowledge they have in their background. Observation gives the designer a chance to verify the accuracy of the questionnaires and interviews. They can observe the users in action, notice their habits, work styles, and strengths.

This information is usually gathered by a Business Analyst ot Human Factors Engineer. Analysts usually don't gather much information on the users. However, it is important to document both the users' backgrounds and how they physically perform their jobs.

Task Analysis

Now that the users are profiled, the jobs they perform need to be understood as well. In particular:

1. What tasks do the users perform and in what order are they performed?
2. How long do these tasks take to complete?
3. Which tasks are most frequent?
4. Which tasks are urgent?

These questions will again provide information for the design guide and during the design phase. Understanding the tasks and task order will identify how information will flow through the system, and from screen to screen. Tasks that are frequent or urgent should be easily accessible and easy to complete. Do not confuse this information with traditional functional analysis. Functional analysis identifies the details of the business process, much of which

happens behind the scenes. The goal of task analysis is to gain an understanding of how the users perform their daily jobs.

Task analysis has two parts -- what users do and how they do it. The first part is covered by the Functional Analysis which is usually performed by Business Analysts. The Functional Analysis describes exactly what the users need to do to perform their jobs. How users perform these functions can be discovered during user observation. Do not confuse the "what" with the "how." Functional Analysis may discover that users need to enter information in a log after they make a phone call to a customer. However, user observation may discover that users actually take notes in a notebook, and enter these notes in the log at the end of the day. Knowing how the users perform their jobs will facilitate an interface design that caters to their needs, rather than managements misconceptions.

User Interface Design Guide

Now that all of this information has been gathered, it needs to be digested and made available to those actually designing and developing the user interface. It is not enough to simply repeat the information to the designers; it must be processed, in context, and conclusions must be drawn from this information. For example, if it is discovered that the users will have 15 inch monitors, windows can not be bigger than 1024x768 pixels. If users currently use paper notebooks to perform their tasks, then a notebook metaphor can be used for the application. This type of information, and not raw data, will be most helpful to designers. The resulting document is often called a "User Interface Design Guide."

A design guide should contain three sections: project background, design guidelines, and appendices. Project Background contains information about the users, tasks, and general project assumptions. Begin with a thorough profile of the users, including job descriptions, computer experince, hardware, and software. This will allow the designers to gain an understanding of the end users and the user's environment. A desciption of the tasks to be performed comes next. As in the design methodology, the users and tasks are the cornerstone of the design guide. Information on the tasks themselves, as well as the order and flow of the tasks shold be covered. The section should conclude by making assumptions about the development environment and design approach. Development tools should be chosen, as should any commercial or custom control packages, including version numbers. A metaphor, if appropriate, should also be identified here.

Design guidelines must be customized for each project. All widgets should be discussed here, as well as window frames, accelerator keys, standard dialogs, and screen layout. Standards such as sizes, labeling, coloring, menus and toolbars, icons, dialogs, and other special design considerations should be discussed in detail. It is important to be specific in this section, as loose guidelines will likely produce the same result as no guidelines, broken standards and the loss of a common look and feel.

The appendices should contain any supporting documentation obtained during user and task analysis. This could include any paper forms currently in use, data descriptions, hardware/software inventories, and user questinoaire results. In addition, a list and description of available widgets should be included. This could be an actual tool palette that designers can pull controls off of, printed for the document. By creating a palette of controls, all designers will be using the same version of a control, and can easily select which controls to use. Finally, a vocabulary of terms to be used on the interface and a list of error messages should be included. The vocabulary section should include both words and icons and their meanings and uses. This is especially important if the users have a common language, such as the financial world's language with phrases like "return on investment," "guarantor," "pre-claim," and "compound interest."

Design guidelines are best written by someone with a thorough understanding of Human Factors issues and principles. If a Human Factors Engineer is not available, Business Analysts and Developers can create a design guide together. The Analysts will have the information about the users and tasks, and Developers will understand the technical limitations and existing standards, such as Microsoft standards.

A common mistake is to create a design guide, and just let it sit on designers' desks. This guide needs to be enforced through design reviews and check-ups. Also remember that design guidelines need to be specific to the project; it is unlikely that one design guide will apply to many projects.

User Interface Design

Once user and task analysis has been completed, and the information processed and documented, the iterative design process can begin. The next three stages -- design, prototype, and test -- will likely melt together into one. Design requires prototyping for visualization of the design, and a prototype is useful only for testing, and the results from testing will fuel a new design.

This cycle can take place in a day, or over months, and can repeat as many times as necessary to create a satisfactory design. What defines a satisfactory design differs with each organization, and each project. Some may require a sign-off from the users; others may establish specific metrics such as "Users complete order entry in 60 seconds with 2% errors." A combination of these two goals will create an interface that the users like, while still satisfying the business requirements of speed and accuracy. Neither should be forgotten.

User interface designers must keep in mind the standards set forth in the design guide. As the design progresses, it should be reviewed by other designers, or a design committee to ensure it meets the standards.

The design process should include input from users, developers, and management. Whenever a new project is undertaken, there is resistence at some level. By including everyone in the design process, everyone has some control over their own fate. Without early buy-in from users, developers, and management, expect delays later as everyone voices their objections during roll-out.

Iterative Prototyping

The purpose of prototyping is to put the design in a form that can be viewed, manipulated, and tested. Prototypes can be in many forms. Low-fidelity prototypes can be simple paper-based or computer-based drawings. Medium-fidelity prototypes can be pseudo-applications that show some of the functionality, often using canned data or arranged scripts. High-fidelity prototypes look like real applications, and perform almost all of the functionality of the final product. Medium and high-fidelity prototypes are often created using tools such as Visual Basic or HyperCard.

The type of prototype used will often depend on the complexity of the interface, the time available, and the skills of the designers. Medium and high-fidelity prototypes that closely mimic the final product will likely provide a more accurate test for users, but the time required to build the prototype will limit the number of iterations that can be completed. Low-fidelity prototypes will allow many quick iterations. Typically, low-fidelity prototypes are used in early iterations to test general look and feel, and overall navigation. As the iterations progress, medium-fidelity prototypes give a better feel of the potential final product.

Be careful to avoid some common pitfalls of prototyping**.** First, **use the same controls in prototyping that you will be using in development.** For example, while many applications are written in Visual C++, prototypes are often done in Visual Basic. VB has a larger set of controls than VC. Do not build a prototype in VB that uses controls not available in VC. Second, prototypes that use canned data and scripts respond faster than applications that use database access. Users that see the speedy prototypes will be disappointed with the speed of the final product. In the prototype, **add a delay where processing should occur to give a realistic feel for the speed of the final application.**

Usability Testing

Usabillity testing comes in many flavors. Formal usability testing requires a dedicated room with hidden video cameras, software that records user activity, microphones, detailed test scripts, and established metrics. However, usability testing can be as easy as a computer, a user, and an observer.

The real goal of usability testing is simply to watch a user interact with the software. It is important to get multiple opinions on the usability of the interface. Since everyone has different skills, attitudes, and work habits, it is important to see how well a variety of users can work with the interface. Do they understand the interface? Can they perform their tasks easily? Do they keep making errors? Ideally, usability testing should include the actual users of the product; however, using other developers, novice users, or similar users is better than no testing at all.

What's Next

Incorporating usability methods into a design effort requires understanding the underlying principles of user interface design, following a methodology that puts user in the center of the design, and developing and enforcing a set of design standards. This document discusses the first two requirements. Once these have been covered, a user interface design guide must be developed. The guide must give detailed explanations of the controls, vocabulary, interaction, and metaphors used in the design effort. Once the guide is created, and the design effort has begun, the standards must be enforced through design reviews, prototype testing, the creation of re-usable components.