Payoff

The evolution of Web technologies as a legitimate business tool has generated expectations that Web applications will perform equally to, if not better than, common commercial software. At the same time, the newness of the technologies has generated numerous misconceptions and abundant misinformation on interface design for Web applications. This article’s recommendations for attaining usable Web-based application interfaces aim to help users successfully apply Web-based software to accomplish their work and save organizations time and money in lost productivity, training, and technology rejection or abandonment.

Problems Addressed

Until recently, most Web-based development was conducted in an atmosphere largely forgiving and tolerant of the shortcomings associated with World Wide Web technologies and their interfaces. Most organizations had only one or two knowledgeable Web developers who were rarely challenged to meet standards comparable to those for traditional software applications. Web sites providing unique information or services encountered little or no competition from similar sites. Although there was no shortage of enthusiasts, expectations were generally low and interest was sustained merely by the Web’s novelty and potential. Similarly, the relatively small number of total potential users meant that any gains or losses resulting from a high-quality or poorly designed Web application were mostly small, if not nonexistent.

As Web technologies have matured, much has changed. Today, the business operations of many corporations hinge on the effectiveness and efficiency of their intranets, and the development of Web marketing strategies is a must. In more and more cases, Web technology is being used to provide resources that greatly enhance the capacity to conduct work. Furthermore, the availability of immediate cross-platform solutions and the relative ease of applications development make Web-based applications the most practical software solution for many organizations.

The evolution of Web technologies from a plaything to a legitimate business tool has generated expectations that Web applications will perform equally to, if not better than, common commercial software. Over the past 10 years, the commercial software business has been driven by usability and the need to provide users with an intuitive, error-tolerant, efficient, and productive user interface. This should not have come as any surprise, because it is fairly obvious that software that allows more work to be done, better and faster, is far preferable to less usable software. The same relationship between usability and business success also exists regarding Web user interfaces.

Usability Basics

In the late 1980s, an interesting phenomenon began to occur in the commercial software business as marketing people and materials started to tout software as user friendly, based on implementation of a graphical user interface (GUI). Because a GUI was thought to be the singular ingredient necessary for attaining user friendliness, often little else was done for the sake of usability.
This phenomenon has left the term user friendly so overused and misused that it no longer has much meaning. Just as GUIs were once touted as the one element needed to achieve usability, today, a Web front end is often cited as the basis for asserting the usability of Web-based products. As was the case for GUIs in the 1980s, these claims typically go unchallenged, and organizations purchase too many products without requesting proof of their usability.

How is usability attained? There are many approaches, and the one chosen depends on the importance of the software. If software is to be frequently used by large numbers of people or for critical activities or both, the approach followed should provide a high level of assurance that usability is attained. Such an approach may include job analysis and task analysis, followed by rapid prototyping interspersed with frequent usability testing. In contrast, if there will be limited numbers of users and the software and its functions are noncritical, it is often sufficient for the developer simply to adhere to design guidelines and obtain informal peer reviews. The importance of software usually varies between these two extremes, so the level of effort devoted to ensuring usability is varied as well.

Cost Justification

Common Misconceptions

The most common misconception about usability among a broad range of software developers and engineers from various disciplines is that usability is largely common sense. Were common sense and good intentions all that is necessary to consistently produce usable software products, the term computer anxiety would never have become part of today’s common vernacular, companies would not have to commit substantial sums to training, help desks, and other user support services, and there would not be such a high incidence of companies investing capital to computerize their business processes only to achieve marginal improvements in productivity.

The misconception that usability is only common sense does not only promote the delusion that usability can be attained at no expense, it also perpetuates the misapplication of talented software developers to the job of user interface design. This task distracts these individuals from the challenges of code development and without the necessary know-how is only rarely a source of great rewards.

Usability comes at a cost. Most corporate software development projects involve significant technical challenges that consume the bulk of both financial and human resources. To commit to usable design requires that resources be diverted from the core code development to various activities associated with the definition, prototyping, testing, and refinement of the user interface. Furthermore, a usable interface often imposes requirements on the supporting software that are inconvenient and met only by overcoming certain difficulty. The net result is that software, Web-based or otherwise, with a highly intuitive user interface is likely to cost more to develop than if usability were neglected.

The tradeoff that results from weighing the importance of the software against the level of effort devoted to development of the user interface increases in importance as corporations automate their business practices and pursue information-driven processes. Usability costs must be paid whether they are incurred through up-front development costs, training costs, lost productivity, or technology rejection and abandonment. The most cost-effective approach for applications that will be used frequently by a large number of users and involve operations for which errors may have costly and otherwise undesirable
consequences is almost always to shift costs to development and make the investments necessary to ensure usability.

Case Study

The case of a hypothetical Web-based time-card application demonstrates this costing logic. The application is to be used once per week (a conservative estimate given that most employers encourage staff to make time-card entries daily, if not more frequently) by 8,000 employees. Experience with many such Web-based applications makes it reasonable to assert that thorough involvement of usability specialists in the development of the time card will reduce the average transaction time from four to three minutes.

This reduction results from improving the efficiency of transactions and reducing the likelihood of errors. Such improvements may be gained in many different ways. For example, integration of two pages into one avoids lost time as users wait for pages to download across busy networks. Similarly, providing immediate access to supporting information such as time-charging codes could yield equivalent savings by avoiding the need for users to go to another location and search for necessary information. Finally, presenting text and data in a readable format can easily prevent errors in data entry, reducing the likelihood of an erroneous time card from 5% to less than 1%.

Presuming an average cost for employees of $50 per hour, the following savings are calculated: 1 min/employee * 8,000 employees = 8,000 min

\[
\begin{array}{c|c}
\text{50$/hour} & \text{= $0.83/min} \\
\text{60 min/hour} & \\
\end{array}
\]

\[
\begin{array}{c|c}
\text{50$/hour} & \text{= $0.83/min} \\
\text{60 min/hour} & \\
\end{array}
\]

$0.83/min * 8,000 min = $6,640/week $6,640/week * 52 weeks/year = $345,280/year $345,280/year * 5 year life span = $1,726,400 savings

These savings only address the cumulative sum of lost productivity. These same improvements to the user interface would also serve to reduce the time required for training, regardless of whether training occurs formally or informally. Although it should be possible to develop an interface that is sufficiently intuitive to obviate the need for training, for the present purposes a reduction in average training time from 30 to 15 minutes is assumed. Such a reduction leads to the following additional savings: 15 min/employee * 8,000 employees = 120,000 min $0.83/min * 120,000 min = $99,600 savings

Still, the savings do not stop here. Typically, this type of application necessitates some type of user support, whether formally through a help desk or informally through co-workers answering each other's questions. Once again, it is reasonable to assert that improvements to usability would result in a reduction in the number of employees who seek assistance on at least one occasion from 15% to 5%. Without even considering that some employees may not only seek assistance, but seek assistance on multiple occasions, further savings are calculated as follows: 10% fewer employees seeking assistance = 800 employees Average time spent on assistance = 15 min * 2 employees = 30 min 30 min * 800 employees = 24,000 min $0.83 * 24,000 min = $19,920 savings
In this example, the savings realized from a 10% reduction in the number of first time requests for assistance should cover a good portion if not all of the costs incurred from including a usability specialist on the interface design team and conducting usability testing. Granted, accommodating usability within the software design may also increase software development costs because the most usable design is often not the easiest to develop. These costs, however, should also be dwarfed by the sum of the potential savings.

The decision to design for usability is always a matter of cost transfer. Designing for usability transfers cost to development. When usability is neglected, either intentionally or inadvertently, a decision is made to transfer costs to training and to the overhead charged to nearly every account to which employees allocate their time. The example of the Web-based time-card application illustrates that by paying a little more up front to ensure usability and transferring costs to development, substantial savings are realized over the life span of the software. By paying a little extra for usability, a buyer purchases quality and the long-range savings that accrue from it.

The Web Design Challenge

A plethora of sources offer guidance on interface design for Web-based applications. These sources include Web sites offering style guides or other design recommendations, books addressing Web page design, and Web developers willing to share their opinions. Given the newness of the Web and the relatively rapid evolution in the capabilities afforded by HTML and Web browsers, the amount of so-called expertise being offered on good and bad design practices is astounding.

A recent study comparing Web design guidelines to traditional human-computer interface (HCI) guidelines suggests the need for caution in adopting Web guidelines regardless of their source. The study of 21 Web style guides found that the time devoted to Web design guidelines was less than one-twenty-fifth that devoted to traditional HCI guidelines. Furthermore, 75% of the more than 350 distinct design recommendations found appeared in only one style guide. This suggests that most of the advice being offered is merely personal opinion. The fact that only 20% of the 270 Web-relevant recommendations found in traditional HCI style guides appeared in any of the Web style guides suggests that existing, readily available knowledge concerning user interface design is largely ignored.

The study also asked a group of human factors practitioners to rate the importance of each of the 270 Web-relevant recommendations found in the traditional design guides to the usability of an interface. Of the 20 recommendations rated most essential to usability, only one was found in any of the Web design guides. Thus, Web design guides do not only fail to address much of what has traditionally been accepted as effective user interface design practice, they also fail to consider those facets of design most essential to usability.

It is generally recommended that developers of Web-based applications approach various sources of design guidance with due skepticism and follow instead the rich body of knowledge relating to user interface design, including guidelines, evaluation, and testing. This well-researched, widely accepted, and generally reliable source of Web-relevant guidance contains the collective knowledge of hundreds of practitioners derived from countless hours of laboratory and field research concerning facets of user interface design that either contribute or distract from usability.
Attaining Usable Web-Based Application Interfaces

The surest formula for attaining usable Web-based application interfaces is to follow a process that incorporates the identification and resolution of usability concerns into every phase of the interface design. This process involves up-front analysis to gain an understanding of the user and the job.

There is no more frequently cited heuristic within human factors than “Know thy user and you are not thy user.” In essence, this heuristic instructs interface designers to make no assumptions regarding what the user needs, prefers, understands, and can use. Most important, just because software may seem easy and make sense to the development team does not mean that the user will be able to understand and use it.

As the interface is developed, designers should follow established guidelines regarding usability design. Lastly, nothing should be taken for granted. Thorough usability testing should be conducted with representative users applying the software, or reasonable prototypes, to perform tasks representative of those for which the software is intended.

Although usability is largely achieved through the process followed in developing the user interface, a great deal of knowledge regarding the facets of design contributes or distracts from an interface's usability. The following sections present some of the HCI guidelines judged most essential to Web usability by the group of human factors experts in the previously discussed study. The guidelines illustrate the effects that specific user interface design decisions may have on the ability of users to successfully apply software to accomplish their work.

Direct Usability of Information

Information should be presented in a directly usable format that does not require decoding, interpretation, or calculation. This means that users should not be given an answer that requires them to seek other references for its interpretation. Although incorporating various reference documents into an application may greatly expand the scope of the application, such additional labor is typically paid for many times over through the resulting increased productivity. Furthermore, not incorporating these references assumes users both possess and know how to use them.

Similarly, users should not be required to use such items as calculators for operations than an application can be programmed to perform. This is because of the probability of human error resulting from reading values from a screen and entering those values by a keypad or keyboard.

Ease of Navigation through Data Displays

When displayed data exceeds a single display, users must have an easy mechanism for moving back and forth through material. The most common violation of this design principle regarding Web-based applications occurs when reports are written from a database directly into an HTML file and the resulting table extends beyond the right border of the browser window. The likelihood of error is introduced because users must awkwardly scroll back and forth and may lose track of the row from which they are reading. In these cases, the costs of thousands of users scrolling back and forth and losing their places within data reports must be weighed against the additional effort required to reformat the reports for display with minimal or no scrolling.
Concise Instruction Levels

Users should not be expected to read more than three help displays or remember more than five points. This statement from traditional HCI sources may be too lenient for Web-based applications because, strangely, users exhibit a much lower tolerance for written instructions presented through Web interfaces than through traditional computer text displays. Thus, for Web-based applications, the requirement for concise, to-the-point help is considerably more stringent and requires greater attention to the content of help systems than might be expected.

Consistent Use of Color

Color should be used consistently within a display or across a set of displays because misinterpretations result from the fact that users both intentionally and unintentionally attribute meaning to colors. Unfortunately, developers often choose colors for aesthetic reasons and fail to consider the unintended meanings users assign to them. Throughout the design of an interface, therefore, the developer must assume that users will assign meaning to colors and exercise care in their selection and use.

Distinct Naming

Similar names for different functions should be avoided. This is because in choosing items from a menu, pull-down list, or table, especially with familiar interfaces, users often discriminate solely on the basis of the general outline of the selection without actually reading the words. For this reason, the labels placed on buttons, in menus, or as list items should be distinct, both semantically and with regard to the visual appearance of the words themselves.

Indication of Function Actuation

A positive indication of function actuation should be provided. This guideline has particular ramifications for Web-based applications. In many respects, Web browsers provide the subtlest of cues that user actions are having an effect. The user cannot be assumed to recognize that a selection has been made on the basis of flying meteorites or spinning electrons that appear against a small graphic within the browser window.

For these reasons, other cues that a user action is having an effect should be provided. For example, most Web users are familiar with the concept of pages wherein text loads before images. Similarly, for back-end processes requiring that the user wait, it is often possible to provide a message informing the user that a request has been received and is being processed. Likewise, for long, perhaps graphically intense downloads, the user may be provided with a confirmation message that the requested download is about to commence.

Confirmation of Destructive Actions

Users should be required to confirm potentially destructive actions before the action is executed. Although such confirmation requests are a standard feature in most non-Web user interfaces, this design practice has often been neglected with software developed for the Web. This situation is exacerbated by the ease with which Web developers may insert a
Clear Form button that all too frequently is placed immediately adjacent to buttons used for frequent operations such as Submit.

The common assertion that confirmation messages are a nuisance is usually made by experienced, frequent computer users and rarely by novice or infrequent users. Unless designing for a highly competent user population that has expressed its disfavor with confirmation messages, the developer should always err conservatively and include them.

**Efficient Data Input**

Data input actions should minimize user actions and memory load on the user. In short, any transformations, formatting, or similar modifications of user input should be done by the computer. It is unproductive for users to expend their resources performing various data manipulations to prepare data to be inputted into the application when these manipulations could be done by the machine.

**Logical Data Entry Formats**

Data entry formats should match source document formats. When a Web interface is used in transferring data from paper forms to electronic data files, the interface should mimic the paper form in sequence and layout as closely as possible.

**Automated Data Entry**

Data should be automatically entered in a field when it is known or can be computed. Once again, overall productivity is served when writing a few extra lines of code to perform an operation or filling in data accessible from one or more data bases results in the transfer of work from the user to the machine. Similarly, if the user has entered data once, it should be filled in later and not require reentry by the user.

**Simplified Data Entry Rules**

Complex rules for entering data should be avoided. In general, flexibility should be the rule. For a social security number, for example, the user should be allowed but not required to use dashes. For a phone number, users should be allowed to enter the area code, but if they do not it should be determined by the machine based on the three-number prefix.

For every data input, therefore, consideration should be given to the various formats by which a user might naturally enter the data. To the extent allowed, all such formats should be accommodated by stripping away excess punctuation and referring to translation tables or other similar mechanisms.

**Required and Optional Field Differentiation**

Cues should be given to distinguish required from optional fields. Many applications request information that is useful, but not necessary. Because users often assume every space must be filled, they devote inordinate amounts of time to searching for information, the benefit of which is far outweighed by the cost of its retrieval. Thus, unless it is truly necessary, users should have the option of skipping fields for which they do not readily know or cannot readily obtain the desired information.
Feedback on Data Input

The user should be provided positive feedback regarding the acceptance or rejection of data input. Many applications allow users to submit requests that will not be filled for several minutes or hours. In such cases, the user should be provided feedback regarding the acceptance or rejection of the request on submission and not when the results are returned. In particular, this calls for routines that validate the user request prior to its submission for processing.

Error Correction

An easy mechanism should be provided for correcting erroneous entries. This mechanism should make it easy for the user to identify and correct any errors made. Should a request be rejected, the user should not be required to reformulate the request from the beginning, but be allowed instead to correct only the erroneous entry and resubmit the request. Similarly, the application should provide sufficient details regarding the location and nature of errors so that correction is intuitive and easily accomplished.

Recommended Course of Action

No one intentionally designs interfaces to be nonusable. At the same time, usability does not come without some expenditure. Consequently, in environments in which development costs and schedules drive design decisions, it is far too easy to neglect usability or assign it low priority. Too often user interfaces are slapped together after all other functional elements are essentially completed. Similarly, design issues that are critical to the usability of the product are all too often relegated to the status of “nice to have.”

It is interesting that a software bug that causes a 1% failure rate will receive endless hours of attention and result in the software ultimately being rejected. Yet an interface feature that causes users to fail to successfully complete 10% of their transactions with the software is considered trivial and not worthy of the precious time and resources required to correct the problem. Similarly, days of analysis, research, and testing are devoted to issues related to functional elements of the software code, but decisions regarding interface features that could be critical to the users’ success or failure are made off-the-cuff, with little or no discussion and rarely any analysis or testing.

There is no shortage of excuses for failing to give usability its due consideration. In the end, however, the costs of this failure are paid through incremental drains on the overall productivity of the enterprise. Just because these costs are largely hidden does not mean they do not warrant correction. By following the guidelines presented in this article, organizations take the first basic steps toward attaining productive and usable Web-based application interfaces that ultimately aid, not hinder, business success.

Author Biographies

Chris Forsythe

Chris Forsythe is senior member of the technical staff in the statistics and human factors department at Sandia National Laboratories in Albuquerque NM.