Fundamental Usability Guidelines for User Interface Design

Conference Paper · June 2008

DOI: 10.1109/ICCSA.2008.45 · Source: IEEE Xplore

4 authors:

Ali Sajedi
University of Alberta
35 PUBLICATIONS **98** CITATIONS
SEE PROFILE

Mehregan Mahdavi
Victoria University Sydney
62 PUBLICATIONS **171** CITATIONS
SEE PROFILE

Amir Pourshirmohammadi
2 PUBLICATIONS **17** CITATIONS
SEE PROFILE

Minoo Monajjemi Nejad
Politecnico di Torino
1 PUBLICATION **17** CITATIONS
SEE PROFILE

Fundamental Usability Guidelines for User Interface Design

Ali Sajedi[1], Mehregan Mahdavi[2], Amir pour shir mohammadi[3] and Minoo Monajjemi nejad[4]
[1]*Department of Computer Engineering, Azad University of Lahijan, IRAN,*
Sajedi@iau-lahijan.ac.ir
[2]*Department of Computer Engineering, Faculty of Engineering, University of Guilan, IRAN,*
mahdavi@guilan.ac.ir
[3]*Department of Computer Engineering, Azad University of Lahijan, IRAN,*
Amir_pourShirMohamadi@yahoo.com
[4]*Department of Computer Engineering, Azad University of Lahijan, IRAN,*
Minoo_libertarian2@yahoo.com

Abstract

Efficient programs are characterized by several parameters including the User Interface Design (UID). From the end-user's point of view, the user interface is the representative of the program. Therefore, friendlier software with limited capabilities is viewed to be more useable than a comprehensive software; in other words, the UI has a great impact on the software to choose.

There has been a great amount of work on UID guidelines. In this paper, we introduce the fundamental guidelines that a designer should consider to increase usability. We consider new aspects including user access, language selection and other technical options in the forms.

The subjects are studied independently from the application and are applicable to all kinds of environments such as web based, desktop, and embedded software. Following these guidelines results in a software which is friendlier, easier to understand and use, more reusable and less tedious.

1. Introduction

It has not been a long time since we had to type instructions in an apathetic environment. We couldn't even imagine that people with just a little information about computers can simply communicate with computers just by clicking mouse buttons without memorizing complicated commands (which that time was accomplished usually by professional users). Nobody thought that one day these people select their desired software by themselves based on their appearance (i.e., UI).

Fast improvement of computer and software lead to new systems; users' choices in these systems not only aren't limited to a set of special commands, but also include a variety of hardware and software tools. These systems usually have complicated interfaces.

User interface (UI) design is the design of computers, applications, mobile communication devices and web sites with the focus on the user's interaction and experience [14], [8], [1].

Typing or scrolling with the keyboard, clicking, double clicking, moving the mouse pointer or it's scroll, dragging the mouse, moving the joystick pad or clicking it's buttons you are directly interacting with the computer; Furthermore, we usually use more devices to interact with the computer. Light pen, bar code reader, scanner, printer and microphone are examples of these devices. UI design is involved in a wide range of projects from computer systems to cars and commercial planes; all these projects require some unique skills and knowledge for the end-user [2], [18].

Software engineers concentrated on the separate steps in development process such as requirements analysis, analysis, design, implementation and test. An important section in design step is the UI design. According to [14], designing the UI needs to accomplish the following cases:

- Functional requirements specification
- Determining the information architecture (e.g., a site map for the web sites)
- Usability testing (by the actual users)
- Collaborate with the graphical designer for building the GUI (Graphical User Interface)

The UI design deals with accurate drawing of shapes on the screen and recognizing the position of interface tools (e.g., mouse pointer). These actions are usually time consuming and repetitive. Thus, the UI design packages were employed. According to the reports,

978-0-7695-3243-1/08 $25.00 © 2008 IEEE
DOI 10.1109/ICCSA.2008.45

Mac App system was one of the first UI design packages that lead to an 80% reduce in coding time.
The main benefits of using UI design tools are as follows [6]:

- Increasing design speed
- Simplicity of changes
- Concentrating on essential details of program rather than controlling the interface
- Uniform design
- …

Nowadays, software interfaces are usually produced by UI design tools. This action even is permeated to web pages. Web designers are always seeking ways to increase the number of their visitors. UI of the web sites should be well-designed to realize this goal. For this reason, some questions should be answered such as "which places of page are suitable for putting links?" or "which part of the site causes the users leave it?" [1]. Answering these questions are classified in the usability of web sites. The mentioned study is usually done through logical evaluation or interview.

2. Related work

Recently, eye-tracking technique monitors users' behaviors while interacting with the computer interface. Eye-tracking measures visual attention as people navigate through web sites. It is useful in quantifying which sections of a web page are read, glanced at or skipped [1], [5]. Web based usability tools such as Web VIP, WET and Web Quilt focus on logging mouse click interactions, but in [1] they focused on mouse browsing paths within a web page.
Also in [7], some applications are introduced for the foot movements. They've built a hardware in which the user could select the items by stepping his/her foot on a limited items on the ground. An example of their UI applications is checking mail.
User Centered Design (UCD) is a process in which the needs, wants and limitations of the user are given extensive attention at each stage of the design [13], [3]. UCD not only requires designers to analyze how users are likely to use an interface, but also requires validity test of their assumptions with regards to user behavior in real world with actual users.
UCD focuses on how people can, want or need to work rather than forcing them to interact with the system in a predefined manner.
In fact, effective applications perform a maximum of work while requiring a minimum of information from the users [15].

3. UI Design Usability Guidelines

In this section, we study the most important UI design guidelines in two main parts: We briefly consider the existing guidelines in the first section. In the following sections, we make valuable suggestions and improvements in the existing guidelines.

3.1. A Survey on UI Design Usability Guidelines

Here, we address the most important principles for UI design. In each case, there are also some suggestions offered.

3.1.1. Consistency

Consistency in UI enables users to build an accurate mental model of the way it works, and this mental model will lead to lower training and support costs [18].
The system should use clear words and commands as a standard based on the platform in whole system, especially if the system consists of several subsystems. For example, it is better to put buttons in consistent places on all windows, use the same wording labels and consistent color scheme throughout.

3.1.2. Flexibility and efficiency of use / user

An expert user should interact with the system as easily as a novice user, but in a different manner. Both users should be satisfied with the same system.
Look at the user's productivity, not of the computers. People cost a lot more money than machines; note the following example;
Which of the following takes less time for the user? Heating water in a microwave for one minute and ten seconds or heating it for one minute and eleven seconds?
From the standpoint of the microwave, one minute and ten seconds is the obviously correct answer. From the standpoint of the user of the microwave, one minute and eleven seconds is faster. Why? Because in the first case, the user must press the one key twice, then visually locate the zero key, move the finger into place over it, and press it once. In the second case, the user just presses the same key–the one key–three times. It typically takes more than one second to acquire the zero key. Hence, the water is heated faster when it is "cooked" longer!
Other factors beyond speed make the 111 solution more efficient. Seeking out a different key not only

takes time, it requires a fairly high level of cognitive processing [15].

Reduce the user's waiting time as much as possible. Note that in an organization, the overall efficiency of everyone should be maximized, not the efficiency of a single person or group. The main efficiency breakthrough in each system is in its fundamental architecture, not the surface design of the UI; thus, try to design a complete system with suitable forms and satisfying relations; in other words, first, design a set of well marked roads, then let the users drive inside. Use brevity and logical grouping in texts and titles (and also in help and documentation) as much as possible. For example, compare the two guidelines for creating an invoice;

A:

- Specify a correct customer number
- Input item code
- Input item name
- Determine the unit of measure
- Fix the discount percentage

B:

Enter:

- Customer#
- Item#
- Item name
- Unit of measure
- Discount (%)

It is obvious that the case B is more abbreviated, yet easier to understand, even though by a non-expert user [2].

Let us consider a real example; you can see the abbreviated form of titles and comments in MS word against sort of explanatory items in SPSS. MS word is friendlier from this point of view, yet for the novice user.

3.1.3. Using colors

Color is a powerful tool, but we should be careful about using it. Use colors as a secondary cue. The mental standards differ from person to person and context to context. Some people do not have colorful screens yet. Furthermore, about 10% of males and a little percentage of females have some form of color blindness. More importantly, the user is not in the ideal situation to see the color as they are because of reflex of light, incorrect degree between eyes and monitor, hardware malfunction and so on. Instead, the designer can focus on graphic, text labels (using different fonts and/or characteristics such as bold, italic, and underline) to imply the difference between different items.

You should use limited amount of colors in each form (at most 4 or 5 colors). The colors should be compatible (with the same law in all forms). It's better to design according to monochromic information of the form (without paying attention to the component colors) and then add colors to increase contrast (color doesn't determine the place of components on the form; nevertheless, the component itself and the relationship with other components determine its place).

Due to not only optical illusion of people in determining colors, but also lack of a unified law for using colors in different science it's better not to use color as an special meaning and use only for speed up in assessment of functionalities.

Also notice in combining colors. For example, red text in blue background makes the eye tired. For using color in your application, you need to ensure that your screens are still readable .The best way to do this is to follow the contrast rule: use dark text on light backgrounds and light text on dark backgrounds. Also attend to the psychology of colors and studying the impact of colors on each other.

3.1.4. Reduce latency

Push latency in the background. Avoid long visual or aural acknowledging against button clicks and so on. Animate the hourglass for actions which take place between 0.5 to 2 seconds so that the users know the system hasn't died. On the other hand, show the estimation of the waiting time for longer actions (more than 2 seconds). This is performed using an up-to-dated text message to inform and entertain users while they are waiting for long processes, such as "to be completed" or an animated progress indicator. For large actions (more than 10 seconds) use a large visual indicator and a beep notifying the user to continue his / her interaction.

Due to the fact that internet is slow, people usually tend to press the same button repeatedly and make the things to be even slower so you'd better trap multiple clicks of the same button or object. You can also make the things faster by removing any element of the application which is not helping.

3.1.5. Metaphors

Good metaphors are stories, creating visible pictures in the mind. Metaphors usually evoke the familiar, but often add a new twist [15]. For example MS windows has an object called Recycle bin. Like a real-world recycle bin and its purpose is something like recycle bins do in everyday life. Desktop metaphor and tape

deck metaphor (seen on many audio and video player programs) are other common examples. Using metaphors we should consider some important factors. For example once a metaphor is chosen, it should be spread, rather than used once at a specific point. The other point is that we should use metaphors which are familiar to everyone because some of them don't cross cultural boundaries well. Metaphor isn't always necessary .In many cases the natural function of the software itself are easier to comprehend.

3.1.6. Help and documentation

The help should be focused on the user's task and needs concrete steps to be carried out, but shouldn't be too large. For example we can use "help browser", "tool tips" or other kinds of context-sensitive help. We can also use "wizards" which guide the user through the step-by-step process, etc. There should be a global search capability for searching a word in the total system's information, a command in the total commands and describing the steps of each operation. Many of these solutions are used commonly by many applications today.

3.1.7. Simplicity along with perfectly

To avoid complexity, first, make simple but complete forms without extra functionalities. Then contrive suitable links and communications to each other [4]. Users usually consider various subjects and make the form so busy. By putting only related subjects in one form and then embedding suitable links between them avoid complexity.

3.1.8. Explorable interfaces

Make actions reversible; in other words, sometimes people want to find out what happens if they choose an action. They don't really want to execute the action. Thus, using undo-like operations in the forms is desirable, although it is hard to implement and keep track of by the designers and programmers [1].
Allowing a way out is another situation, although it is easier to stay in.
According to Fitts' law [9], the time to acquire a target is a function of the distance to and size of the target with direct relation with the distance and indirect relation with size of the target; $t \sim \log(d/s)$ where t is time to acquire a target, d is distance and s is size of the target. It is proven that Macintosh's pull-down menu acquisition is approximately five times faster than Windows. This is because of getting in people's way (sometimes unnecessarily) in windows taskbar.
Fitts' law indicates that the four corners of the screen are the most quickly accessible targets in the computer displays. After the corners, the four sides (top, button, left and right) are the next. Use them for more important objects.

3.1.9. Match between the system and the real world

The system should use concepts familiar to the user, rather than system-oriented terms. In UI design, you should exploit your originality and imagination; for example, in MS Windows, deleting an item is done with dragging it into the trash. It is difficult to find such a fascinated match in the life.
A friend said: "when I wanted to buy a portable phone, I tested many characteristics such as memory and sound effects, but I had special attention to the size of 'talk' button"! This is straightforward in designing really friendly interfaces; in other words, one should see the main task of a form in a particular way.
We can see another fascinated mapping in [17]; when you put the items inside your pocket, you usually put cash and wallet in one pocket and the keys in the other one. This mapping is used when you are categorizing items in the form; however, the categories can follow an accorded standard.

3.2. Main Factors of Evaluating a UI

The main factors of UI design for a regular program are as follows:

1- Time to learn (average time for a user to learn to interact with the system)
2- Rate of errors by users (average number of errors produced by a user or type of users)
3- Retention over time (how much can a user retain his / her knowledge about working with the system after a period)
4- User satisfaction (this parameter can be determined with questionnaire or conversation)

We want to introduce useful guidelines to create more friendlier interfaces considering these factors. Our suggestions are commonly in the context of some general guidelines.

3.3. Improving Usability Guidelines

Here, we suggest some useful improvements in UI design guidelines.

3.3.1. Access control

The designer should know the current user, his/her goals, skills, experience and needs in order to make the UI better [19], [2]. Using this information we can create an interface that helps users achieve their goals. Controlling users' access in these systems is so important.
In database based applications, first, several users are defined in the database. Then, their permission to the related tables and views is defined. In many cases, an inner layer is applied to handle the many to many relationships between the user and the permissions [10], [11]. This layer is named role, after determining the access of roles to the Database elements, the permissions of users is applied by relating them to the roles.
The above approach has all the benefits of DBMS control, but two defects:
First, the produced error messages are originated from the DBMS, although they are refined and customized by exception handling commands.
Second, this approach is data driven not functionality driven. So there can be transaction failures due to access deny.
Our aim is to propose a functionality driven approach. The functionality driven control is closer to the end-user; hence can be assigned to the end-user easily. The control is applied before triggering functionalities not in the progress of a transaction; thus saving the user time and reducing computer load.
There can be also a better sight on the accessibilities in this approach.
We imply that all controls should be applied in the form of access to the forms, menu options, even buttons and other controls. This can be controlled by first, saving the users and all functionality triggering facilities (e.g., forms, menu options, even buttons and shortcut keys) and then controlling their relationships in the application by means of hiding or disabling the access ways. By this way, the end-user only sees the options with full access as usual, the options with only read access as disabled / read only and the prohibited controls are hid automatically, so the user doesn't mix up by prohibited option that run incompletely.

3.3.2. User control and freedom

User's characteristics should be considered in design. For example, UI of a day care center should include big colorful icons with just bit of words, but an operator, librarian or a professional programmer need their own special UI.
As we know in the case of some software, there may be many different kinds of users so it may be useful to make a list of user dichotomies, such as "skilled vs. unskilled", etc. We can also talk to some real users.
The users should be informed about what is going on. Attend to the Human Interface Objects. They are not necessarily the same as Object Oriented objects. Human interface Objects include documents, folders, shortcuts, menus, buttons and so on. They have a standard way of interacting and resulting behavior [16], [3].
We suggest determining the system states or at least its basic state. The "basic state" is the main state that the user can start any transaction from there. It differs from the main form of an application; in fact, in each form the basic state is a stable status that can be origin of every transaction in that from.
Our purpose is to give control to the user by defining the basic state and using it in confusing situations. In these cases, the user can enter this state without producing errors. Note that sometimes the previous state is a lateral state the user doesn't want to complete it. In fact, the design should be intelligent enough so that the system automatically cancels any incomplete tasks and enters the basic state.
For example, in a translator, usually an edit / combo box, one or more areas for the translation(s) and several options are in the main form. The main state in this form is when the active control is the edit / combo box and the cursor blinks in it. The user can switch between the mentioned widgets by pressing the "tab" key. But pressing the defined key (e.g., F3) enters directly to the basic state and can type words in the edit / combo box instead of confusing in determining the current active control or at least losing rather long time pressing the tab key repeatedly.
Also the users should have an "emergency exit" in every unwanted state, without having to go through an extended dialogue.

3.3.3. Minimize the user's memory load

The system should memorize the options, actions and conditions from previous parts rather than the user. The information should be filled out automatically in the current form in the correct positions. In case of decisions, the decision making process should be done according to the current set of information and related dependencies to the previous information.
The least need to technical information should be considered; thus, doing important actions by non expert users. The 4GL languages [12] were initiated from this idea. For example, consider a non expert user (with common capabilities of interacting with the computer) creates a new network by plugging the two sockets of the network cable and follows the top down

consecutive guides without knowing about IP addresses.
Note that this is more than a UI design concept; In fact, only a comprehensive insight over the issue can result in a very high level framework to be used in the context. Hence, all the desired functionalities should be analyzed profoundly to design an exhaustive framework and interface. Having this framework, the only problem is about the user's knowledge about the context itself not the way of performing the task.
For example, consider multilingual forms; up to now, there is a global language variable that is used in all the forms. If we change it, the typing language of all forms is also changed.
An intelligent approach is to have a separate language variable for each form / view. A more intelligent approach is to save the language status in every place / component in the last entrance to it. This can be done by saving the language status in its corresponding variable in each component; In fact, some changes are inevitable in the ancestors of the components in components hierarchy.

3.3.4. Creating multilingual forms

A challenge in designing multilingual forms for a UI designer is to have the widgets arranged properly when changing the language. For example, in a two language form (e.g., English / Persian), the place, alignment and size of widgets should be determined based on the default language and the embedded texts. At first, the separate approach seems to be successful, but as the forms being more complicated, keeping copies of the information in separate layouts becomes inefficient; besides, it has redundancy in updating forms.
A more comprehensive approach is to handle the issues in multilingual forms by means of inheritance along with embedding some events in the current forms. In fact, a child form class should be designed to handle alignment and size issues. This class inherits properties and events of all current forms. All cases should be designed dynamically (i.e., all actions should be done for all components of the form). Moreover, it needs to use a separate (text) file to contain all text labels of each form in each language. The new forms are derived from this form, hence from the initial visual form and contain all common properties, events and methods.

3.3.5. Minimalist design and aesthetic

The design should be ordered and well-formed. Extra information should be hid; however, there is a direct relation between relevancy of the items and their visibility. Design and selection of tasks and assigning commands for triggering them is an important activity of the designer [16]. For example, in a text editor, we have several triggering styles for a set of tasks [6]:

- Simple and most frequent tasks like "delete" and "enter" are triggered with a simple key
- Less frequent tasks like "copy", "paste" and "print" are triggered with a combination of keys (like Ctrl + C or Ctrl + P)
- Infrequent tasks like "auto correct" or "grammar checking" are triggered using a menu (and possibly entrance of the conditions in the related form)

Sometimes it is needed to do an action in a form without changing the status or entering in another form. For example, in a movie player in the full screen mode, the designer should embed some widgets and components in the proper places to issue some important commands such us changing the volume, forward and backward without exiting full screen. In these cases, the designer should consider a tradeoff between minimalist design and ease of access.
This, in the first place, needs accurate examination of the needs of each form in terms of tasks or functionalities.

3.3.6. Error prevention / handling

The designer should prevent users from going into an error prone state by an exhaustive control. When an error occurs, the users should never lose their work. Commit / rollback commands can be used to prevent lose of changes.
Addresses, functions, and table and variable names should not be used in the error messages. The problem should be indicated precisely and at least one solution should be suggested.
This is rather more than determining some error messages to be shown in appropriate places. The structure needs essential changes and refinements to handle all exceptions correctly such as indicating log files, temporary tables / files and appropriate commands to embed in the underlying code. The code is referred to as the solution of the error and can be triggered by the end-user or admin of the system.

3.3.7. Anticipation

Using menu as a simple but complete reference is recommended. A new user can easily learn the assignments of different parts of the program using its menu. The most important subject about designing menu is to organize the items properly (especially if

the choices are so many); in other words, you should use categorizing menu for two or more levels.
Bring to the user all the information and tools needed for each step of the process. Do not expect user to gather information or provide necessary tools. The user wants to do an action as rapidly as possible. The first step to do the action is triggering that action in a proper manner. Therefore, it is required to have an exhaustive control over the top layer of the UI to find the places and ways of placing widgets and components to choose from. In other words, the end-user should see the required choices whenever and wherever needed. This can be done by means of enabling some disabled widgets and components.

4. Conclusion

We studied guidelines for designing a good UI. We also made suggestions for improving the quality of the UI design. We discussed a number of parameters that determine the quality of a UI. The effect of our suggestions in improving the UI quality based on these parameters is evaluated and presented in Table 1.

Table 1. The effect of guidelines on the important factors of a UI; H = High, M = Middle, L = Low

	Time to learn	Rate of errors by user	Retention over time	User Satisfaction
Access control	M	H	L	H
User control & freedom	M	H	L	H
Minimizing the user's memory load	M	M	M	H
Creating multilingual forms	M	M	L	H
Minimalist design and aesthetic	M	L	L	M
Error prevention / handling	L	H	L	M
Anticipation	H	L	M	H

These guidelines are independent from the environment and application. Following these guidelines, can result in designing and implementing a successful interface for a wide range of software (e.g., commercial, real time, scientific, Artificial Intelligence, web based, desktop, embedded, and system software).

5. Future work

The UI of desktop applications differ s from web based applications. Some of these differences are considered in the literature. The UI design metrics is yet unclear; thus, defining usability, being friendly, expandability, understandability, testability and other metrics are useful activities especially for web design. These issues are left for further work.

6. Acknowledgements

We would like to thank Ms. Zahra Zebardast for her constructive comments and editorial revisions.

7. References

[1] E. Arroyo, T. Selker, and W. Wei, "Teaching User Interface Design using a web-based Usability Tool", *CHI 2006*, ACM, Montreal-Canada, April 22-27, 2006.
[2] H. Störrle, "Group Exercises for the Design and Validation of Graphical User Interfaces", *Modellierung'2002,* Tutzing, GI Lecture Notes in Information, 26.3.2002, pp. 12.
[3] A. Gaffar, N. Moha, and A. Seffah, "User-Centered Design Practices Management and Communication", *Proceedings of HCII 2005,* Human Computer Interaction International, Las Vegas, Nevada, USA, 2005.
[4] P.J. Molina, and H. Trætteberg, "Analysis & Design of Model Based User Interfaces", *computer science,* Springer Netherlands, 2005, pp. 211-222.
[5] D.L. Scapin, J. Vanderdonckt, C. Farenc, R. Bastide, CH. Bastien, C. Leulier, C. Mariage, P. Palanque, "Transfering knowledge of User Interfaces Guidelines to the Web, Tools for Working with Guidelines", *Proceedings of the International Workshop on Tools for Working with Guidelines TFWWG'2000*, Springer- Verlag, London, 2000.
[6] P.Medina, J. Luis, D. Chessa, Sophie, and A. Front, "A Survey of Model Driven Engineering Tools for User Interface Design", *computer science*, Springer, Berlin, Heidelberg, 2007, pp. 84-97
[7] B. Meyers, A.J. Brush, S. Drucker, M.A. Smith, and M. Czerwinski, "Dance Your Work Away: Exploring Step User Interfaces" *CHI '06 extended abstracts on Human factors in computing systems***,** Association for Computing Machinery, Inc, April 2006, pp. 387-392.
[8] A.H. Dutoit, T. Wolf, B. Paech, L. Borner, and J. Rückert, "Using Rationale for Software Engineering Education", *Proceedings of the 18th Conference on Software Engineering Education & Training(CSEE&T 2005),* IEEE Computer Society, 18-20 April 2005, pp. 129-136.
[9] P.M. Fitts, "The information capacity of the human motor system in controlling the amplitude of Movement", *Journal of Experimental Psychology,* 1954, pp. 381–391.
[10] R. S. Sandhu, E. J. Coyne, H. L. Feinstein and C. Youman. "E. Role-Based Access Control Models". *IEEE Computer*, 29, Feb 1996.

[11] P. Lamb, R. Power, G. Walker, M. Compton, "Role-Based Access Control for e-Service Integration", LNCS Volume 3729, Springer, 2005, pp. 816 – 828
[12] J. M. Verner and G. Tate, "Estimating size and effort in fourth-generation development". *IEEE Software,* July 1988.
[13] Http://en.wikipedia.org/wiki/User_centered_design
[14] Http://en.wikipedia.org/wiki/User_interface_design
[15] http://www.asktog.com/basics/firstPrinciples.html
[16] Http://www.useit.com/papers/heuristic/heuristic_list.html
[17] http://www.zenhaiku.com/archives/usability_applied_to_life.html
[18] http://www.ambysoft.com/essays/userInterfaceDesign.html
[19] http://www.sylvantech.com/~talin/projects/ui_design.html