Using Principles to Support Usability in Interactive Systems

Mauricio Lopez
Dept. of Computer Science and Engineering
York University
Toronto, Ontario, Canada M3J1V6
malchevic@msn.com

ABSTRACT
This paper discusses the use of general principles that are used to promote usability in interactive systems. These interactive systems include some commercially software applications, computing devices, and interfaces. First, a summary of those principles is provided. Then, six detailed examples from the chosen application(s), device(s), and interface(s) are discussed, two for each principle. For each two, one demonstrates the correct application of the principle, and the other the incorrect application of that principle. Screen snaps and other visual aids are used to support and organize the examples.

Author Keywords
Usability, principles, critique, design

ACM Classification Keywords
Evaluation/methodology

INTRODUCTION
Increasing and promoting usability in interactive systems is an important goal in the human-computer interaction field. By usability we refer to the level of ease with which people can employ a particular device in order to achieve a particular goal. In the human-computer interaction field there are some design rules that are used in order to attain and increase usability. The most abstract design rules are called principles. Dix [1] divides these principles in three main categories: Learnability, flexibility, and robustness. For each of these categories, there is a further set of principles that affect them. This paper consists of two parts: the first part provides a summary of all the principles and the second part criticizes the use of this principles in common applications such as software, devices, and interfaces. Six details examples are provided, two for each principle. For each two, one shows the correct application of the principle, and the other the incorrect application of that principle.

SUMMARY OF PRINCIPLES
As it was mentioned before, all the general principles are first divided in three main categories: Learnability, flexibility, and robustness. Each category contains more specific principles. These specific principles will support each category. The first category deals with learnability.

Learnability
Learnability simply refers to the capability of an interactive system to enable the user to learn how to use it initially, and then exploit all the features that it offers with relative ease. There are five specific principles that support learnability:

Predictability
The desire of most people during their lives is to get what they expected according to their actions. Usually this process starts by planning and choosing an action that will satisfy a certain goal. Most of the time people show confidence about the future outcome of that action. The source of that confidence is the knowledge that we have gathered during all our lifetime about how the things work. In other words, we can predict the outcome of our actions based on our current knowledge. The same concept can be applied to the interaction between a human and an interactive system. An interactive system possesses predictability if it offers and helps the user to understand and predict the consequences of a certain action. With this principle, the user must be able to choose a certain action in the interactive system and be confident that this action will work correctly and will give the expected outcome.

Synthesizability
After the user executes an action to satisfy a certain goal, it is critical that the interactive system provides some sort of feedback. Usually there are different ways to represent this feedback, but all of them must provide the user with effective and reliable information about the effect of the action after its execution and the changes that it performed. This ability is called honesty. Honesty can happen immediately or eventually.

Familiarity
As it was mentioned in the definition of the predictability principle, people usually choose a certain action based on the knowledge that they have gathered through all their lives. Several interactive systems such as software applications are designed in such a way that their goal is to help the user to create an analogy between an object or situation and the computer application. The users will be more familiar with that application because they will be
able to apply the knowledge that they have about the object or situation to the computer application.

Some objects can also have physical properties that give us information about how they can be used. These properties are called affordances and they can increase the level of familiarity. It is important that the interactive systems possess affordances because they will decrease the cognitive friction that must of the computer systems have.

**Flexibility**

*Dialog initiative*

All interactive systems obviously need some type of interaction between the system and the user to achieve something meaningful. This interaction can be considered as a conversation between two persons. One important issue is to decide who has the control or initiative in this conversation. The ideal solution in the point of view of the user is to have freedom from the system imposed constraints. This means that the user must be able to abandon, suspend, or resume any activity at any point. This form of dialog between the system and the user is called user pre-emptive. The dialog is called system pre-emptive if the system has the initiative and asks the user for information, usually in the form of modal dialog boxes. When interactive systems are designed, the programmers have to be careful about choosing which type of dialog is going to be needed in order to achieve maximum flexibility. A good solution must combine both types of dialog since it is a bad idea to give the user all the freedom to do whatever he wants to accomplish.

**Multi-threading**

Multi-threading allows the user to do simultaneous tasks. Usually the interactive systems need to support multi-modality, a term which is more related to the human-computer interaction field since the term multi-threading is usually associated with computer architecture and operating systems. There are two types of multi-modality: concurrent and interleaving. Concurrent multi-modality allows the user to do many activities or tasks at the same time but in different places, for example, the user can have many windows with different applications running, like text editor, p2p, and a music player. Interleaving multi-modality implies that two tasks seem to happen at the same time and in the same place.

**Task migratability**

It is always dangerous to leave people doing routine tasks. Usually they get bored and stop concentrating on the task. This increases the probability of making errors, some of them fatal. That is where the task migratability principle is needed. Task migratability measures how easily a certain task can be moved between user and system. Routine tasks can be automated, thus saving time and decreasing the probability of making errors. Like the other principles, the best thing to do is to allow constant switching between the user and system, since some tasks cannot be leave completely to the discretion of the system.

**Substitutivity**

This term simply means allowing equivalent values of input and output to be substituted for each other. An example is the flexibility of changing Celsius to Fahrenheit degrees.

**Customizability**

Customizability is what many users desire. They want the interface to adapt to their needs and customs. This modification can be either done by the user (adaptability) or by the system (adaptivity). Some of the most common customizations that are required include providing choice of methods, allowing short-cuts and permit users to change features.

**Robustness**

*Observability*

The user must be able to figure out what is the current state by simply having a quick snapshot at the interface. The interface must give useful and intuitive information to the user. Some basic questions must be answer easily: where am I? Where am I going? Where have I been? What can I do now?

**Recoverability**

Undoing and recover from errors is a critical feature that must be present in interactive systems. The system must give the user the ability to take correction action once an error has occurred. Also, the user should be able to go back to any previous point, this is called reachability. One critical part that is often ignored by some programmers is that the errors messages must be concise, informative, specific, and constructive. This allows the user to understand better what the problem was and recover from it quickly.

**Responsiveness**

In order to keep a sense of stability, the response time of the system must be fast, otherwise the user will be confused. Also, the system must issue some sort of notification after an action has been requested. In this way, the user will know that the system is taking care of the task.
LEARNABILITY EXAMPLES

Example 1: Correct application of familiarity principle in the Wii remote controller

Tech products, frequently referred as gadgets, are an excellent source to find examples of learnability. In particular, the principle of familiarity is very easy and sometimes very obvious to find in these devices. In this example we will focus on the affordances that the Wii remote, the controller of the brand new videogame console of Nintendo, has. Affordances are important since they provide strong clues to the operation of things [3]. When they are used correctly, the user usually will figure out what to do by merely inspecting the device or object. There have been many products with very poor design that use incorrectly the principle of affordance throughout the short history of high-tech devices. The website Cnet [2] has a top ten list of the worst high-tech devices, as notable examples we can mention the Apple Hockey Puck mouse, the Gizmondo, and the Squircle. These devices have in common the lack of usability due to the bad design and lack of affordances. The user cannot figure out what is the purpose or even how to operate them by mere inspection. Norman argues that when things need pictures, labels, or instructions, the design has failed [3]. On the other side, the Wii remote offers an excellent example of the correct usage of affordances.

Example 2: Incorrect application of synthesizability principle in the CSE Webmail

Figure 1 shows the Wii remote, which consists of two parts: the controller itself and the Nunchuk controller. The controller has a one-handed design unlike traditional controllers; this design promotes a notion to the user that it must be used for pointing. Figure 3 shows how the design promotes usability by exploiting the affordance principle. The users can figure out easily how to grab the controller because it resembles a stick. Due to the design of the controller, the fingers are placed in an intuitively and smooth manner. The Nunchuk controller also offers a very good design, the controller is contoured to perfectly fit the user hand and its usage can be discovered by simple observation. The Wii-remote was designed as simple as possible since the goal of Nintendo was to attract the non-gaming community.
We defined synthesizability as the user’s ability to determine the effect of future interactions in part one. Immediate honesty is a critical principle that helps the successful execution of this ability. The user must be able to see immediately any change that was performed and receive some feedback from the computer. This feedback notifies the user about the change so no further interaction is needed. The CSE Webmail system lacks of honesty in one of its features. After the user sent a message, no notification is given by the system. It is hard to figure out if the mail was sent successfully. Norman states that when an action has no apparent result, you can conclude that the action was ineffective [2]. In order to confirm that the mail was sent correctly, the user must select the folders option, and then inspect the contents of the sent-mail folder. This is an example of how a small omission of immediate honesty can affect the usability of a whole program. By simply issuing a confirmation message, further interaction of the user can be avoided.

**Figure 4. Apple Iphone interface.**

**FLEXIBILITY EXAMPLES**

**Example 3: Correct application of dialog initiative and task migrability in the Apple Iphone**

In some devices, it is critical to define if the user or the system has the initiative in the interaction. A user-driven interaction favors flexibility. The Apple Iphone interface is a good example of correct application of dialog initiative. It offers freedom from the system imposed constraints and the user is able to abandon, suspend or resume activities at any point. The touch screen facilitates the flexibility of the interface. By just selecting a particular feature, the user can quickly change from one task to another without problem. Task migrability is also present in the Iphone. For example, if the user is listening to music, the song will fade out when a call is received and will fade back in when the call ends. Let’s analyze the process in this example. First, the user has the control of the system. When the call comes in, the system will take control, fade out the song and pass the control to the user. The user will either answer the call or reject it. When one of these options terminates, the system will take control and will fade back in the song.

**Figure 5. Apple Iphone Notes application.**

**Example 4: Incorrect application of customizability in the Apple Iphone**

Sometimes users like to customize the user interface according to their needs and preferences. They can increase the productivity and promote a better attitude towards the system. The Apple Iphone has a feature called Notes (See figure 5). This feature allows the user to write messages using a virtual QWERTY keyboard. The problem with this feature is that it completely omits customization capabilities. The user cannot change the font, size, and color of the letters. The default settings are unique and there is no option to change them. In this example, the designers chose what they thought was the best option and ignored the possible customization desire of the users.
ROBUSTNESS EXAMPLES

Example 5: Correct application of observability in the Apple Iphone

The Apple Iphone perfectly applies observability. By simply looking at the interface, the user can easily evaluate the internal state of the system. The interface can answer the main questions regarding observability: Where am I? Where am I going? What can I do now? The Iphone offers browsability. The arrangements of the icons and features in the main screen are clear and concise and offer the user a quick snapshot of the tasks that can be done and the current state of the system. The most common features (phone, mail, browser, and Ipod) are clustered together (see Figure 6); this arrangement facilitates the quick assimilation of the features. The Iphone also offers persistence (see Figure 7), if the user missed some calls, the phone icon will have a red circle, indicating that a call was missed. The icon will not change until the user selects the phone option and checks the missed calls feature.

Example 6: Incorrect application of recoverability in Open Office Writer

One of the goals of recoverability is to undo errors. But recoverability must also ensure that the error messages are concise, informative, specific, and constructive. When this paper was being written, I had a problem with the Open Office Writer application. When the document was saved for the first time, the program showed an error message as shown in figure 8. This information of this error was not concise, constructive, or specific; there was no clue to figure out why the program was not working since an output and input problem can have many sources. To make things worse, the file was not save. When I tried to save the file again, the same error appears. The program enters to an infinite loop. Clearly, this is an example of lack of recoverability: the program never recovered from the error and displayed useless information about it. After many hours I discovered that the problem was not having enough disk space. By simply adding a useful message with that information, things would have been easier to solve.

CONCLUSION

Applying usability principles is critical to ensure good design. They encapsulate the basic principles that promote the usability of the products, which is the ultimate goal of the designers. One has to be careful when applying these principles since incorrect usage can lead to many errors and bad interfaces.

REFERENCES

2. CNET 10 worst tech products
   http://crave.cnet.co.uk/gadgets

3. Norman, D. 2002. The design of everyday things,
   Eds. Basic Books, United States.

4. Dix, A. et al. 2004. Human-Computer Interaction,
   Eds. Prentice Hall, United States.