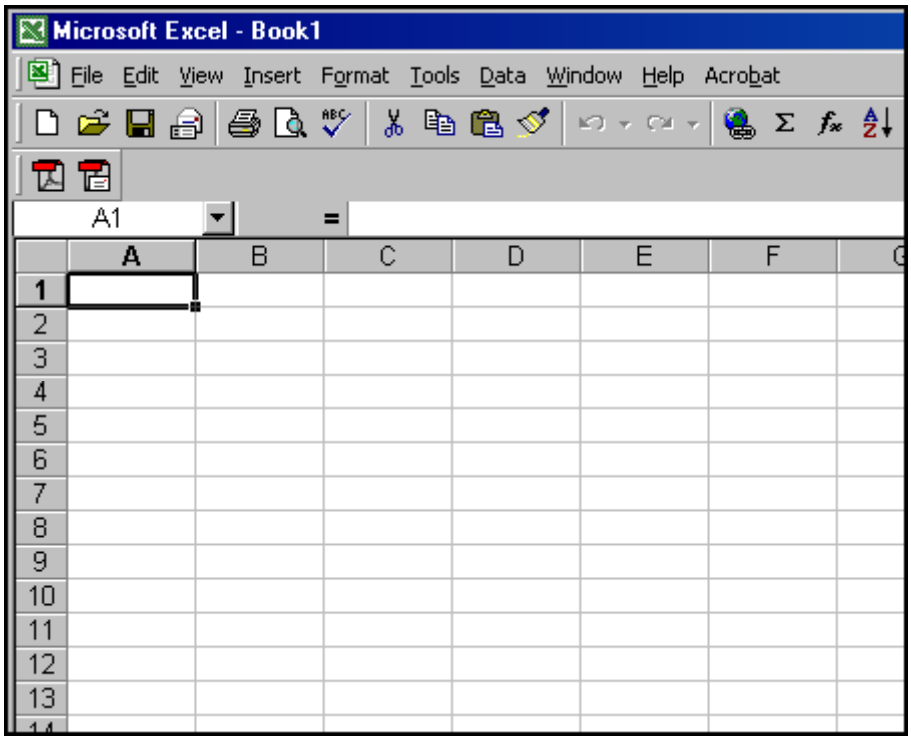
MATCH USER/DESIGNER MODELS



A spreadsheet program modeled as a ledger book

Whether an application/system is for COOPERATIVE WORK or SINGLE-USER WORK, it needs to obey a consistent MODEL; that is, an underlying set of principles or rules by which the application operates.

* * *

One of the most fundamental tasks of the interaction designer is deciding on an application model. He knows that the application he is designing needs to accomplish a certain task. What he doesn't know is, how should the program behave in response to user actions? Can the user enter in text using a keyboard? Does a gesture with the mouse mean something?

In the real world, for any given task, there are probably many, if not infinite, ways of accomplishing it. However, there are some methods that clearly emerge as superior to others.

By extension, the same is true in the computer domain. For any task, there are many programs that will accomplish it. However, there are clearly some programs that are easier to use than others.

Why is this so? Consider an interaction between two humans, A and B. If both A and B share the same conventions of conversation (same language, taking turns, etc.), then they can have a pleasant conversation without any problems. Now suppose A and B come from different cultures. During the course of the conversation, A says something that he considers to be pleasant. B, however, coming from a different culture, regards it as the worst insult imaginable. This misunderstanding happened because A and B *didn't share the same conventions of conversation*.

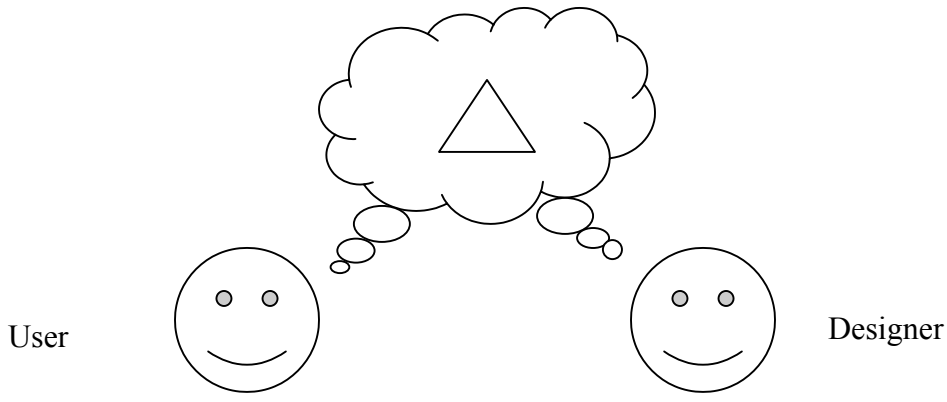
This is precisely the reason why some applications are easier to use than others. When both the user and the application share the same model of behavior, they can cooperatively and successfully accomplish the task at hand. When their models differ, however, many misunderstandings will result, much as in the real-world conversation example.

A classic example of the use of a good model is in spreadsheet programs. Pioneered by VisiCalc, spreadsheet programs today are modeled after real-world ledger books. Because the user has an understanding of how ledger books work, they can extend that knowledge to the unfamiliar interface of a spreadsheet program.

Word processors also use models effectively, specifically the typewriter model. The user types, and words appear on the screen. This may seem obvious, but consider the following alternative design: the user has to click on a virtual keyboard onscreen, letter by letter, to produce words in the document.

Note that models do not necessarily need to have real-world counterparts. Many image programs utilize the painter's model. In this model, once something is drawn onto the screen, it loses its properties as a distinct object, and just becomes part of the canvas, much as a painter works. Adobe Photoshop is an example of an application that uses this model effectively.

Therefore, when designing an application, create a model that is readily comprehensible by the user. It is possible to cause the user to adapt to the application model, if the two models are sufficiently similar and there is good reason for doing so. However, the ideal solution is to investigate the user's understanding of the task, and design the model such that it matches the user's. Doing so will allow the user to comprehend the interaction with minimal effort.



* * *

One possible characteristic for the model to possess is that of DIRECT MANIPULATION. It can also utilize NOUN-VERB CONSTRUCTS. Finally, the interface should have VISIBLE AFFORDANCES, present FEEDBACK, and give the user a sense of CONTEXT.

DIRECT MANIPULATION
Problem: You want to present the objects of interest in an interface, and to allow the user to manipulate the objects without using complex, typed commands
Solution: use a direct manipulation interface, in which the objects of interest are presented on the screen, and can be manipulated through various mouse actions.

NOUN-VERB CONSTRUCTS
Problem: The user wants to execute an action on an object
Solution: let the user choose the object, then choose the action, as opposed to choosing the action and then the object.

VISIBLE AFFORDANCES
Problem: You want to make it clear that certain objects can be acted upon, and what those actions are
Solution: distinguish the manipulable objects through their appearance

FEEDBACK
Problem: You want the user to know that an action has been received
Solution: Using a variety of techniques (progress bars, dialog boxes, etc.), show the user that his command has been received and is being processed.

CONTEXT
Problem: You want the user to not get lost while accomplishing the task
Solution: Constantly give the user a sense of context; let him know where he's been, where he is, and possibly where he is going.