OBJECTIVES

At the completion of this chapter, you will be able to:

- Identify the major goals of user interface design.
- Describe the different types of prototypes.
- Create a paper prototype.
- Design a system interface.
- Use the Backus-Naur Form to create an interface control document.

PRE-TEST QUESTIONS

The answers to these questions are in Appendix A at the end of this manual.

1. What is the primary purpose of the user interface?

.....................................................................................................................................
.....................................................................................................................................

2. In designing a user interface, what are some things that should be considered?

.....................................................................................................................................
.....................................................................................................................................
INTRODUCTION

Object modeling languages are becoming increasingly expressive and complex. The use of a language like UML can increase the abilities of user interface designers for modeling application domains and interactive systems. The recent development of techniques which assist in user interface design during the object-oriented development process provide programmers with the tools necessary to develop robust, user friendly applications. Some of the better techniques include task analysis or preliminary user interface analysis using interface prototyping, which has proven significant in object identification.

USER INTERFACE DESIGN

Designing user interfaces can be difficult. Large development projects will often employ human interface analysts who specialize in designing user interfaces. Smaller projects may have to rely on programmers to design user interfaces. For most programmers, user interface design appears to be a trivial exercise. However, the user interface plays an important role in creating customer satisfaction. A good user interface will be transparent to customers, freeing them to judge the system on its process and output. A poor user interface can be an obstacle to an otherwise well-implemented software system.

A good user interface follows user interface conventions. These may vary between operating systems. For example, Microsoft publishes a manual of user interface design guidelines, which describes the ways in which users expect a Windows application to react. Other environments, such as the World Wide Web, have their own informal conventions. Programmers often assume they understand these conventions, but experience shows that applications do not conform without a well-established process for user interface design.

In this chapter, you will learn to design user interfaces that accommodate ergonomic and accessibility considerations. Later, you will learn to determine an interface design's potential for success through user testing.
User interface principles

User interface design incorporates more than mere aesthetics. A considerable body of empirical data regarding human-computer interaction has been collected. This data has been used to formulate a number of principles at play in modern software user interfaces. These principles are the use of metaphors, self-evident components, consistency, and state visualization.

Metaphor

It is often beneficial to design a user interface as a metaphor of a real-world object. The Windows desktop is a good example. The desktop has folders that lie on top of it, and folders contain documents. If you choose to use a metaphor in developing a user interface, be sure to use it consistently across the entire project. Do not create a metaphor that applies only to a very narrow aspect of the program. Also, realize that while metaphors provide one way to design user interfaces, not all software systems employ metaphors. Thus, a good balance must be struck between the arcane and useless metaphors and a complete lack of designing interfaces that relate to the actual world.

As stated above, the desktop has become a very successful metaphor, and there are other real-world objects that are being implemented for programs. For instance, there is work being done on many platforms to have a file system explorer that is a metaphor for a real storage area. Thus, directories with larger size on the hard disk will appear as larger structures within the program. There are many other metaphors used in user interfaces, but it is important to remember that not every situation will warrant a real-world object metaphor.

Self-evidence

The most critical feature to build into a user interface is self-evidence. The function of various user interface components should be readily apparent to even slightly experienced users. However, in most cases, a large number of users will use a system, and each of these users will approach the system from a different perspective.
Experienced computer users will look for controls that are common among applications. Many Windows applications have a standard Windows toolbar that contains buttons to execute common functionalities, such as opening a new or existing document or saving a document. Experienced users will recognize these icons from other applications and feel confident to explore the features specific to your application.

Novice users are often fearful of making mistakes. They may not recognize these icons. Their hesitance may hinder their ability to use a system, which can leave many powerful and useful aspects of a program unused. Making the user interface self-evident eases this fear and facilitates learning. Figure 9-1 shows the standard Windows toolbar. Some of the icons on the toolbar are intuitive. Most people will understand immediately that clicking on the image of a printer will cause the current document to be printed.

For instance, in the example below, two pieces of paper on top of each other may indicate “Paste” to experienced users, but may mean nothing at all to new users of a system. However, the Paste icon is not threatening and seems to imply that no harm will come to a novice user for using that command. Thus, while the exact meaning of the icon may not be grasped by every user, the knowledge that pressing that icon will not cause serious harm to a program should be evident to all users.

![Library System](image)

**Figure 9-1: Standard Windows toolbar**

**Consistency**

User interfaces should be consistent. There are two types of consistency: internal consistency and external consistency. Internal consistency is consistency within the interface of a single software system. For example, if the library system allows users to access a patron’s account information by either scanning her library card or manually entering her library card number, it should allow users to do the same when checking out a book to a patron. This type of consistency makes learning the system easier and using the system faster.
External consistency is consistency between a piece of software and the underlying operating environment. For example, Macintosh applications open multiple independent document windows, while Microsoft Windows applications usually implement the multiple-document interface in which a single application window contains multiple document windows. Another example of external consistency between different recent versions of Microsoft Windows is the Start Menu. For any Windows user, that user will know that programs can be accessed through the Start Menu for any version of Windows that has been produced since Microsoft Windows 95. This external consistency in the operating system allows for smoother transitions for users when they switch between different Microsoft products.

A well-designed user interface will follow the conventions of the operating environment in which it resides, making sure to use the internal and external consistencies which are already used.

State visualization

In the preceding chapter, you created activity diagrams for each of your top-level use cases. These activity diagrams illustrate the life cycle of a use-case and the various states in which it can exist, from initiation to completion. Often, these states correspond to a request for input from the user. State visualization is the principle that visual cues should be offered to illustrate the transition from one state to another.

For example, when a librarian checks a book out to a patron, he must first scan the patron’s library card. Following this state, the librarian will either be notified that the patron owes an overdue fine, or he will begin scanning the book to be checked out. All of these states should be visually distinct so that the librarian can learn to quickly identify each. If the screen notifying the librarian of an overdue fine looks too similar to the screen requesting the assets to be checked out, the process will be slowed down while the librarian examines the screen to determine the new state.

Figure 9-2 is based on the activity diagram for the Check Out Asset use case. A dialog box alerts the librarian if the patron owes an overdue fine. The screen used to pay an overdue fine is visually different from the screen to used enter the assets being checked out. These visual representations of state allow the librarian to work more efficiently.
State visualization is important, but be sure to avoid creating a modal interface. A modal interface can exist in only one state at a time. For example, a modal interface might restrict a librarian's ability to check out books if he is using the system to perform a time-consuming task, such as generating overdue notices. Modal interfaces can be easy to use, but they are inflexible.

Figure 9-2: State visualization
USER INTERFACE PROTOTYPING

A prototype is a simulation of the actual system. User interface prototypes are used to test the usability of the interface. Essentially, a prototype is a tool for defining and refining requirements. A prototype may take the form of a piece of software that implements the interface and most basic functionality of a system, or it may be as simple as a set of screen shots called a paper prototype. Following are descriptions of several classes of prototypes.

- **Horizontal prototype**: implements a broad set of features and a shallow implementation of each feature.
- **Vertical prototype**: implements a subset of the features with a more thorough implementation.
- **High-fidelity prototype**: a software prototype designed to look and operate in the same manner as the final software system.
- **Low-fidelity prototype**: a non-software prototype that uses diagrams, documents, or presentations to communicate the user interface.

Each of these classes of prototypes can be created using either the rapid or paper prototype process. The two prototype development processes are described below.

Rapid prototyping

Rapid prototyping is a process by which a working demonstration version of a piece of software is created in a very short period of time. When designing a user interface, several rapid software prototypes may be developed and tested with users before a final user interface is agreed upon.

A software prototype is never developed into a fully operational system. The prototype is used only for demonstration purposes. It communicates a vision of the final product's appearance, and it serves as a catalyst for new ideas. Software prototypes are often developed using Rapid Application Development (RAD) techniques. No emphasis is placed on performance. Software prototypes are developed using RAD tools such as Visual Basic or Delphi even if the actual software system will be developed using another language.
Paper prototyping

Paper prototyping is used to communicate the same information as a software prototype without the expense of developing a working piece of software. Paper prototypes use screen shots and storyboards to illustrate the system's appearance and functionality. Paper prototypes may be part of a formal interface requirements document. Figures 9-3 through 9-9 are the interface requirements document for the library system, based on a paper prototype process.

Figure 9-3: Interface Requirements Document—page 1
Introduction
This document describes the user interface for the Library System being developed by Superior Software Corp. for the County Public Library Commission.

The user interface will operate on two platforms: Library personnel will access the system using a Microsoft Windows platform. In order to access the system using VT100 terminals, library personnel will access the system using VT100 terminals. This document describes the personnel interface.

Personnel Interface
This section describes the personnel interface for the Library System. The personnel interface operates using Microsoft Windows.

Screen layout and design
Library personnel will use the Microsoft Windows interface. This section contains instructions for the layout of the interface.

The user interface is divided into four main sections: Check Out, Check In, Accounts, and Home. The Check Out screen allows personnel to check out Library items to patrons. The Check In screen allows personnel to check items in. The Accounts screen allows personnel to enter account information, including fines, fees, and account maintenance. The Home screen allows personnel to add and delete items from the account database, generate overdue notices, and generate invoices and account reports.

Figure 9-4: Interface Requirements Document—page 2
Figure 9-5: Interface Requirements Document— page 3
Interactions styles

This section describes the ways in which users will interact with the library system interface.

The four major sections represent the four primary sections of the interface. The user clicks on one of the four tabs at the top of the screen to select an option. The user can then select between 1.5 and 2.5 when clicking on a different tab. The action will then be processed and the user returns to the original tab.

Check out assets

A librarian can check out an asset to a patron using the Check Out screen. The user will select the option by entering the account number in the Account Number field. The system will then allow the user to select the item using the keyboard or using the check box. The Account Number field is required in order to assign an account number.

Alternatively, the user may select the patron's name or a portion of the name in the Name field. The system will then enter the patron's address as a portion of the address in the Address field. The system will perform a search against the account database for the name or address entered. If a match is found with the account number entered, the user will proceed with a confirmation screen.
Figure 9-7 Interface Requirement Document-page 5
Figure 9-8: Interface Requirements Document—page 6

After a customer has been selected, a list of the account currently checked out by the person will appear in the Account Check Out box. The customer’s current account balance will appear below the Account Balance label. The customer’s account history will be displayed in the Account History list. Figure 9-8 shows an example of viewing a customer account information.

![Account Example](image)

If the account is free, the user may indicate the account by clicking the Pay Free button. When the Pay Free button is clicked, the user is presented with a dialog box indicating the account is free. The account is collected from the customer’s account balance.

The user may also create a new account at the Account screen. Clicking the Create New Account button opens the Create New Account window. The Create New Account window is shown in Figure 9-7.

![Create New Account window](image)

Figure 9-7: Create New Account window
Administer

The user enters an asset number into the Asset Number field. The title of the asset is entered in the Title field. The asset type is selected from the Asset Type list. The user clicks the OK button when the task is complete. A second window appears with additional information about the asset to be added, as provided in the task. The fields in this form vary depending on the asset type.

The user may delete assets from the Add Asset screen. The user can delete assets by clicking the Delete Asset button. Clicking the Delete Asset button presents the user with the Delete Asset window. Figure 9 shows the Delete Asset window.

Figure 9-9: Interface Requirements Document—page 7
Humans are not the only users of a software system. In today's wired world, computer systems must often interface with other computer systems. For example, the library system must be able to communicate with software systems at other libraries when remote users search the asset database. Because this library system must conform to existing communications standards, defining this interface is an important part of the requirements-gathering process.

You can express the syntax of this communication using a metasyntax. The Backus-Naur Form (BNF) is the most widely used metasyntax for expressing context-free grammars. You can use BNF to define the syntax of a programming language, a configuration file, or a communications protocol. Table 9-1 summarizes the Backus-Naur Form.

<table>
<thead>
<tr>
<th>Metasymbol</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;literal word&quot;</td>
<td>Literal words are represented within double quotation marks.</td>
</tr>
<tr>
<td>syntactic_category</td>
<td>Syntactic categories are represented by words without double quotation marks.</td>
</tr>
<tr>
<td>::=</td>
<td>Two colons followed by an equal symbol means &quot;is defined as...&quot;</td>
</tr>
<tr>
<td>[ ]</td>
<td>Items contained within square brackets are optional.</td>
</tr>
<tr>
<td>{ }</td>
<td>Items contained within curly brackets are repeated zero or more times.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 9-1: Backus-Naur Form
The following is an example of a BNF grammar for an external search request:

```
search_request ::= 
  "search"  
  "author = '" author "']       
  "title = '" title "']       
  "isbn = '" isbn "']          
  "date = '" date "']          
  "subject = '" subject "']    
```

The following line is a valid use of the preceding grammar:

```
search author = 'Booch' subject = 'software' subject = 'UML'
```

---

**WHEN THIS IS USEFUL**

Both interface design and prototyping are very useful when developing software. The design of an interface is useful when beginning to put together the way a user will interact with the software. Having users test the interface, allowing for changes that will possibly have to be made, receiving feedback from users and developers on the user interface, all of these are useful to the programmer(s) of a piece of software. Interface design provides ways for the programming team to receive input and make changes early in the design process that will save time and money for a company in addition to making an overall better product.

Interface prototyping is a very important part of the interface design process because it actually allows users and developers to study the interface being implemented and test the interface for problems or unforeseen complications. Without prototyping there would not be a way for the interface design process to receive feedback and testing, which would nullify the entire purpose of interface design.
Exercise 9-1: Defining interface requirements

1. Using a pencil and paper, create rough sketches of the user interface to the grocery store inventory system you created in the preceding exercises.

2. The User Interface Components.jpg file is located in the <CDROM:><Supplemental_Files>Object_Oriented\Analysis_Design\Chapter 9 folder. This file contains basic user interface components that you can manipulate using Paint Shop Pro (or another image-editing application) to demonstrate your user interface. Download Paint Shop Pro from the Jasc Software site at the following URL:
   
   http://www.paintshoppro.com/

3. Create user interface diagrams; you can use the components provided in the User Interface Components.jpg file if you wish. Import your diagrams into Microsoft Word (or another word-processing application) to create a user interface design document.

4. Using the Backus-Naur Form, create an interface control document that contains the grammars for communication between the grocery store inventory system and the hand-held computers used by employees shelving groceries.

SUMMARY

A good user interface is transparent to the user. Poorly designed user interfaces can be a significant obstacle to users. It is often beneficial to design user interfaces as a metaphor of some real-world object. User interface components should be self-evident, as well as both internally and externally consistent. A well-designed user interface will provide visual cues of the software's current state. Interface prototypes provide tools for defining and refining requirements. Software prototypes implement the interface and basic functionality in a simple software application. Paper prototypes utilize non-software tools to describe a user interface. The interface between a software system and external software systems can be defined using the Backus-Naur Form.
POST-TEST QUESTIONS

The answers to these questions are in Appendix A at the end of this manual.

1. Describe how the term “metaphor” is used in interface design.

..................................................................................................................................
..................................................................................................................................

2. Describe the two types of software interface consistency.

..................................................................................................................................
..................................................................................................................................

3. Contrast Vertical and Horizontal prototyping.

..................................................................................................................................
..................................................................................................................................