**UNIT-III  
CHAPTER-III  
USER INTERFACE DESIGN**

The user interface portion of a software product is responsible for all interactions with the user. Almost every software product has a user interface (can you think of a software product that does not have any user interface?). In the early days of computer, no software product had any user interface.

**CHARACTERISTICS OF A GOOD USER INTERFACE**

How to develop user interfaces, it is important to identify the different characteristics that are usually desired of a good user interface. Unless we know what exactly is expected of a good user interface, we cannot possibly design one.

* **Speed of learning:** A good user interface should be easy to learn. Speed of learning is hampered by complex syntax and semantics of the command issue procedures. A good user interface should not require its users to memories commands. Neither should the user be asked to remember information from one screen to another while performing various tasks using the interface. Besides, the following three issues are crucial to enhance the speed of learning:
* **Use of metaphors**1 **and intuitive command names:** Speed of learning an interface is greatly facilitated if these are based on some day-to-day real-life examples or some physical objects with which the users are familiar with. The abstractions of real-life objects or concepts used in user interface design are called *metaphors*. If the user interface of a text editor uses concepts similar to the tools used by a writer for text editing such as cutting lines and paragraphs and pasting it at other places, users can immediately relate to it.
* Another popular metaphor is a shopping cart. Everyone knows how a shopping cart is used to make choices while purchasing items in a supermarket. If a user interface uses the shopping cart metaphor for designing the interaction style for a situation where similar types of choices have to be made, then the users can easily understand and learn to use the interface. Also, learning is facilitated by intuitive command names and symbolic command issue procedures.
* **Consistency:** Once, a user learns about a command, he should be able to use the similar commands in different circumstances for carrying out similar actions. This makes it easier to learn the interface since the user can extend his knowledge about one part of the interface to the other parts. Thus, the different commands supported by an interface should be consistent.
* **Component-based interface:** Users can learn an interface faster if the interaction style of the interface is very similar to the interface of other applications with which the user is already familiar with. This can be achieved if the interfaces of different applications are developed using some standard user interface components. This, in fact, is the theme of the component-based user interface
* **Speed of use:** Speed of use of a user interface is determined by the time and user effort necessary to initiate and execute different commands. This characteristic of the interface is sometimes referred to as *productivity support* of the interface. It indicates how fast the users can perform their intended tasks. The time and user effort necessary to initiate and execute different commands should be minimal.
* **Speed of recall:** Once users learn how to use an interface, the speed with which they can recall the command issue procedure should be maximized. This characteristic is very important for intermittent users. Speed of recall is improved if the interface is based on some metaphors, symbolic command issue procedures, and intuitive command names.
* **Error prevention:** A good user interface should minimize the scope of committing errors while initiating different commands. The error rate of an interface can be easily determined by monitoring the errors committed by an average user while using the interface. This monitoring can be automated by instrumenting the user interface code with monitoring code which can record the frequency and types of user error and later display the statistics of various kinds of errors committed by different users.
* **Aesthetic and attractive:** A good user interface should be attractive to use. An attractive user interface catches user attention and fancy. In this respect, graphics-based user interfaces have a definite advantage over text-based interfaces.
* **Consistency:** The commands supported by a user interface should be consistent. The basic purpose of consistency is to allow users to generalize the knowledge about aspects of the interface from one part to another. Thus, consistency facilitates speed of learning, speed of recall, and also helps in reduction of error rate.
* **Feedback:** A good user interface must provide feedback to various user actions. Especially, if any user request takes more than few seconds to process, the user should be informed about the state of the processing of his request. In the absence of any response from the computer for a long time, a novice user might even start recovery/shutdown procedures in panic. If required, the user should be periodically informed about the progress made in processing his command.
* **Support for multiple skill levels:** A good user interface should support multiple levels of sophistication of command issue procedure for different categories of users. This is necessary because users with different levels of experience in using an application prefer different types of user interfaces. Experienced users are more concerned about the eﬃciency of the command issue procedure, whereas novice users pay importance to usability aspects. Very cryptic and complex commands discourage a novice, whereas elaborate command sequences make the command issue procedure very slow and therefore put off experienced users
* **Error recovery (undo facility):** While issuing commands, even the expert users can commit errors. Therefore, a good user interface should allow a user to undo a mistake committed by him while using the interface. Users are inconvenienced if they cannot recover from the errors they commit while using a software. If the users cannot recover even from very simple types of errors, they feel irritated, helpless, and out of control.
* **User guidance and on-line help:** Users seek guidance and on-line help when they either forget a command or are unaware of some features of the software. Whenever users need guidance or seek help from the system, they should be provided with appropriate guidance and help.

**BASIC CONCEPTS**

Some basic concepts in user guidance and on-line help system. Next, we examine the concept of a mode-based and a modeless interface and the advantages of a graphical interface.

**User Guidance and On-line Help**

* Users may seek help about the operation of the software any time while using the software. This is provided by the on-line help system. This is different from the guidance and error messages which are flashed automatically without the user asking for them. The guidance messages prompt the user regarding the options he has regarding the next command, and the status of the last command, etc.
* **On-line help system:** Users expect the on-line help messages to be tailored to the context in which they invoke the “help system”. Therefore, a good on-line help system should keep track of what a user is doing while invoking the help system and provide the output message in a context-dependent way. Also, the help messages should be tailored to the user’s experience level.
* **Guidance messages:** The guidance messages should be carefully designed to prompt the user about the next actions he might pursue, the current status of the system, the progress so far made in processing his last command, etc. A good guidance system should have different levels of sophistication for different categories of users. For example, a user using a command language interface might need a different type of guidance compared to a user using a menu or iconic interface (These different types of interfaces are discussed later in this chapter). Also, users should have an option to turn off the detailed messages.
* **Error messages:** Error messages are generated by a system either when the user commits some error or when some errors encountered by the system during processing due to some exceptional conditions, such as out of memory, communication link broken, etc. Users do not like error messages that are either ambiguous or too general such as “invalid input or system error”.
* Error messages should be polite. Error messages should not have associated noise which might embarrass the user. The message should suggest how a given error can be rectified. If appropriate, the user should be given the option of invoking the on-line help system to find out more about the error situation.

**Mode-based *versus* Modeless Interface**

* A *mode* is a state or collection of states in which only a subset of all user interaction tasks can be performed. In a modeless interface, the same set of commands can be invoked at any time during the running of the software. Thus, a modeless interface has only a single mode and all the commands are available all the time during the operation of the software.
* On the other hand, in a mode-based interface, different sets of commands can be invoked depending on the mode in which the system is, i.e., the mode at any instant is determined by the sequence of commands already issued by the user.
* A mode-based interface can be represented using a state transition diagram, where each node of the state transition diagram would represent a mode. Each state of the state transition diagram can be annotated with the commands that are meaningful in that state.

**Graphical User Interface (GUI) *versus* Text-based User Interface**

Let us compare various characteristics of a GUI with those of a text-based user interface:

* In a GUI multiple windows with different information can simultaneously be displayed on the user screen. This is perhaps one of the biggest advantages of GUI over text- based interfaces since the user has the flexibility to simultaneously interact with several related items at any time and can have access to different system information displayed in different windows.
* Iconic information representation and symbolic information manipulation is possible in a GUI. Symbolic information manipulation such as dragging an icon representing a file to a trash for deleting is intuitively very appealing and the user can instantly remember it.
* A GUI usually supports command selection using an attractive and user-friendly menu selection system.
* In a GUI, a pointing device such as a mouse or a light pen can be used for issuing commands. The use of a pointing device increases the eﬃcacy of command issue procedure.
* On the flip side, a GUI requires special terminals with graphics capabilities for running and also requires special input devices such a mouse. On the other hand, a text-based user interface can be implemented even on a cheap alphanumeric display terminal. Graphics terminals are usually much more expensive than alphanumeric terminals.

**TYPES OF USER INTERFACES**

Broadly speaking, user interfaces can be classified into the following three categories:

1. Command language-based interfaces
2. Menu-based interfaces
3. Direct manipulation interfaces

Each of these categories of interfaces has its own characteristic advantages and disadvantages. Therefore, most modern applications use a careful combination of all these three types of user interfaces for implementing the user command repertoire. It is very diﬃcult to come up with a simple set of guidelines as to which parts of the interface should be implemented using what type of interface.

This choice is to a large extent dependent on the experience and discretion of the designer of the interface. However, a study of the basic characteristics and the relative advantages of different types of interfaces would give a fair idea to the designer regarding which commands should be supported using what type of interface.

**Command Language-based Interface**

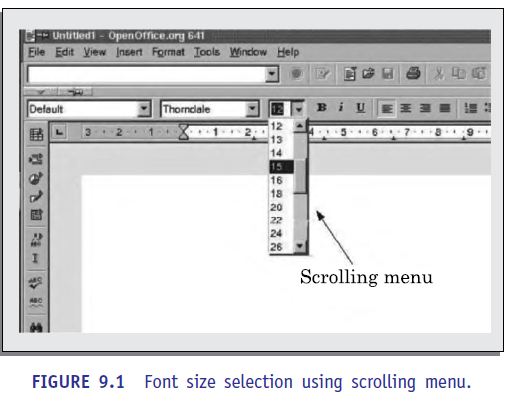
* A command language-based interface—as the name itself suggests, is based on designing a command language which the user can use to issue the commands. The user is expected to frame the appropriate commands in the language and type them appropriately whenever required. A simple command language-based interface might simply assign unique names to the different commands. However, a more sophisticated command language-based interface may allow users to compose complex commands by using a set of primitive commands. Such a facility to compose commands dramatically reduces the number of command names one would have to remember.
* Command language-based interfaces allow fast interaction with the computer and simplify the input of complex commands. Among the three categories of interfaces, the command language interface allows for most eﬃcient command issue procedure requiring minimal typing. Further, a command language-based interface can be implemented even on cheap alphanumeric terminals. Also,
* A command language-based interface is easier to develop compared to a menu-based or a direct-manipulation interface because compiler writing techniques are well developed. One can systematically develop a command language interface by using the standard compiler writing tools Lex and Yacc.
* However, command language-based interfaces suffer from several drawbacks. Usually, command language-based interfaces are diﬃcult to learn and require the user to memorize the set of primitive commands. Also, most users make errors while formulating commands in the command language and also while typing them

**Issues in designing a command language-based interface**

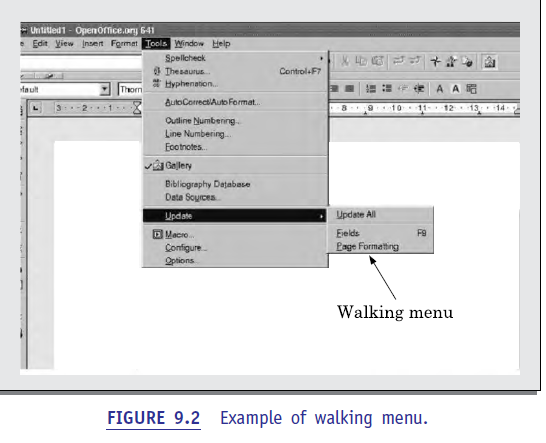
* Two overbearing command design issues are to reduce the number of primitive commands that a user has to remember and to minimize the total typing required. We elaborate these considerations in the following:
* The designer has to decide what mnemonics (command names) to use for the different commands. The designer should try to develop meaningful mnemonics and yet be concise to minimize the amount of typing required. For example, the shortest mnemonic should be assigned to the most frequently used commands.
* The designer has to decide whether the users will be allowed to redefine the command names to suit their own preferences. Letting a user define his own mnemonics for various commands is a useful feature, but it increases the complexity of user interface development.
* The designer has to decide whether it should be possible to compose primitive commands to form more complex commands. A sophisticated command composition facility would require the syntax and semantics of the various command composition
* options to be clearly and unambiguously specified. The ability to combine commands is a powerful facility in the hands of experienced users, but quite unnecessary for inexperienced users.

**Menu-based Interface**

* An important advantage of a menu-based interface over a command language-based interface is that a menu-based interface does not require the users to remember the exact syntax of the commands. A menu-based interface is based on recognition of the command names, rather than recollection. Humans are much better in recognizing something than recollecting it.
* Further, in a menu-based interface the typing effort is minimal as most interactions are carried out through menu selections using a pointing device. This factor is an important consideration for the occasional user who cannot type fast.
* In fact, a major challenge in the design of a menu-based interface is to structure large number of menu choices into manageable forms. In the following, we discuss some of the techniques available to structure a large number of menu items:
* **Scrolling menu:** Sometimes the full choice list is large and cannot be displayed within the menu area, scrolling of the menu items is required. This would enable the user to view and select the menu items that cannot be accommodated on the screen. However, in a scrolling menu all the commands should be highly correlated, so that the user can easily locate a command that he needs.
* This is important since the user cannot see all the commands at any one time. An example situation where a scrolling menu is frequently used is font size selection in a document processor (see Figure 9.1). Here, the user knows that the command list contains only the font sizes that are arranged in some order and he can scroll up or down to find the size he is looking for.



* **Walking menu:** Walking menu is very commonly used to structure a large collection of menu items. In this technique, when a menu item is selected, it causes further menu items to be displayed adjacent to it in a sub-menu.
* A walking menu can successfully be used to structure commands only if there are tens rather than hundreds of choices since each adjacently displayed menu does take up screen space and the total screen area is after all limited.
* **Hierarchical menu:** This type of menu is suitable for small screens with limited display area such as that in mobile phones. In a hierarchical menu, the menu items are organized in a hierarchy or tree structure. Selecting a menu item causes the current menu display to be replaced by an appropriate sub-menu. Thus, in this case, one can consider the menu and its various sub-menu to form a hierarchical tree-like structure.



**Direct Manipulation Interfaces**

Direct manipulation interfaces present the interface to the user in the form of visual models (i.e., icons2 or objects). For this reason, direct manipulation interfaces are sometimes called as *iconic interfaces.* In this type of interface, the user issues commands by performing actions

* on the visual representations of the objects, e.g., pull an icon representing a file into an icon representing a trash box, for deleting the file. Important advantages of iconic interfaces include the fact that the icons can be recognized by the users very easily, and that icons are language-independent. However, experienced users find direct manipulation interfaces very far too.
* Also, it is diﬃcult to give complex commands using a direct manipulation interface. For example, if one has to drag an icon representing the file to a trash box icon for deleting a file, then in order to delete all the files in the directory one has to perform this operation individually for all files—which could be very easily done by issuing a command like delete \*.\*.

**Performing User interface design: Golden rules.**

1. Place the user in control.

2. Reduce the user’s memory load.

3. Make the interface consistent.

These golden rules actually form the basis for a set of user interface design principles that guide this important aspect of software design

**Place the User in Control**

During a requirements-gathering session for a major new information system, a key user was asked about the attributes of the window-oriented graphical interface

Define interaction modes in a way that does not force a user into unnecessary or undesired actions. An interaction mode is the current state of the interface. For example, if spell check is selected in a word-processor menu, the software moves to a spell-checking mode.

**Provide for flexible interaction**. Because different users have different interaction preferences, choices should be provided.

* For example, software might allow a user to interact via keyboard commands, mouse movement, a digitizer pen, a multi touch screen, or voice recognition commands.
* But every action is not amenable to every interaction mechanism

**Allow user interaction to be interruptible and undoable**

* Even when involved in a sequence of actions, the user should be able to interrupt the sequence to do something else (without losing the work that had been done). The user should also be able to “undo” any action.

**Streamline interaction as skill levels advance and allow the interaction to be customized.** Users often find that they perform the same sequence of interactions repeatedly. It is worthwhile to design a “macro” mechanism that enables an advanced user to customize the interface to facilitate interaction.

**Hide technical internals from the casual user.** The user interface should move the user into the virtual world of the application. The user should not be aware of the operating system, file management functions, or other arcane computing technology.

**Design for direct interaction with objects that appear on the screen.** The user feels a sense of control when able to manipulate the objects that are necessary to perform a task in a manner similar to what would occur if the object were a physical thing. For example, an application interface that allows a user to “stretch” an object (scale it in size) is an implementation of direct manipulation.

**Reduce the User’s Memory Load**

The more a user has to remember, the more error-prone the interaction with the system will be. It is for this reason that a well-designed user interface does not tax the user’s memory.

* It defines design principles that enable an interface to reduce the user’s memory load:

**Reduce demand on short-term memory.**

When users are involved in complex tasks, the demand on short-term memory can be significant. The interface should be designed to reduce the requirement to remember past actions, inputs, and results

**Establish meaningful defaults.**

The initial set of defaults should make sense for the average user, but a user should be able to specify individual preferences.

**Define shortcuts that are intuitive.**

When mnemonics are used to accomplish a system function (e.g., alt-P to invoke the print function), the mnemonic should be tied to the action in a way that is easy to remember (e.g., first letter of the task to be invoked).

**The visual layout of the interface should be based on a real-world metaphor.**

For example, a bill payment system should use a checkbook and check register metaphor to guide the user through the bill paying process. This enables the user to rely on well-understood visual cues, rather than memorizing an arcane interaction sequence.

**Disclose information in a progressive fashion.**

The interface should be organized hierarchically. That is, information about a task, an object, or some behavior should be presented first at a high level of abstraction. More detail should be presented after the user indicates interest with a mouse pick

**Make the Interface Consistent**

The interface should present and acquire information in a consistent fashion. This implies that

(1) all visual information is organized according to design rules that are maintained throughout all screen displays,

(2) input mechanisms are constrained to a limited set that is used consistently throughout the application, and

(3) mechanisms for navigating from task to task are consistently defined and implemented.

**Allow the user to put the current task into a meaningful context.**

* Many interfaces implement complex layers of interactions with dozens of screen images. It is important to provide indicators (e.g., window titles, graphical icons, consistent color coding) that enable the user to know the context of the work at hand.

**Maintain consistency across a family of applications.**

* A set of applications (or products) should all implement the same design rules so that consistency is maintained for all interaction

**If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so.**

* Once a particular interactive sequence has become a de facto standard (e.g., the use of alt-S to save a file), the user expects this in every application he encounters. A change (e.g., using alt-S to invoke scaling) will cause confusion.