Module 9

User Interface Design

Lesson 22

Component-Based GUI Development

Version 2 CSE IIT, Kharagpur

Specific Instructional Objectives

At the end of this lesson the student would be able to:

- Explain what is window in terms of GUI.
- Explain what is meant by window management system.
- Identify the responsibilities of a window manager.
- Identify at least eight primary types of window objects.
- Explain what X Window is.
- Explain why the X Window is so popular.
- Explain architecture of an X System.
- Explain what is meant by visual programming.
- Differentiate between a user-centered design and a design by users.
- Explain implications of human cognition capabilities on user interface design.
- Define seven important steps needed to design a GUI methodology.
- Prepare a check list for user interface inspection.

Window

A window is a rectangular area on the screen. A window can be considered to be a virtual screen, in the sense that it provides an interface to the user for carrying out independent activities, e.g. one window can be used for editing a program and another for drawing pictures, etc.

A window can be divided into two parts: client part, and non-client part. The client area makes up the whole of the window, except for the borders and scroll bars. The client area is the area available to a client application for display. The non-client part of the window determines the look and feel of the window. The look and feel defines a basic behavior for all windows, such as creating, moving, resizing, and iconifying the windows. A basic window with its different parts is shown in fig. 9.6.

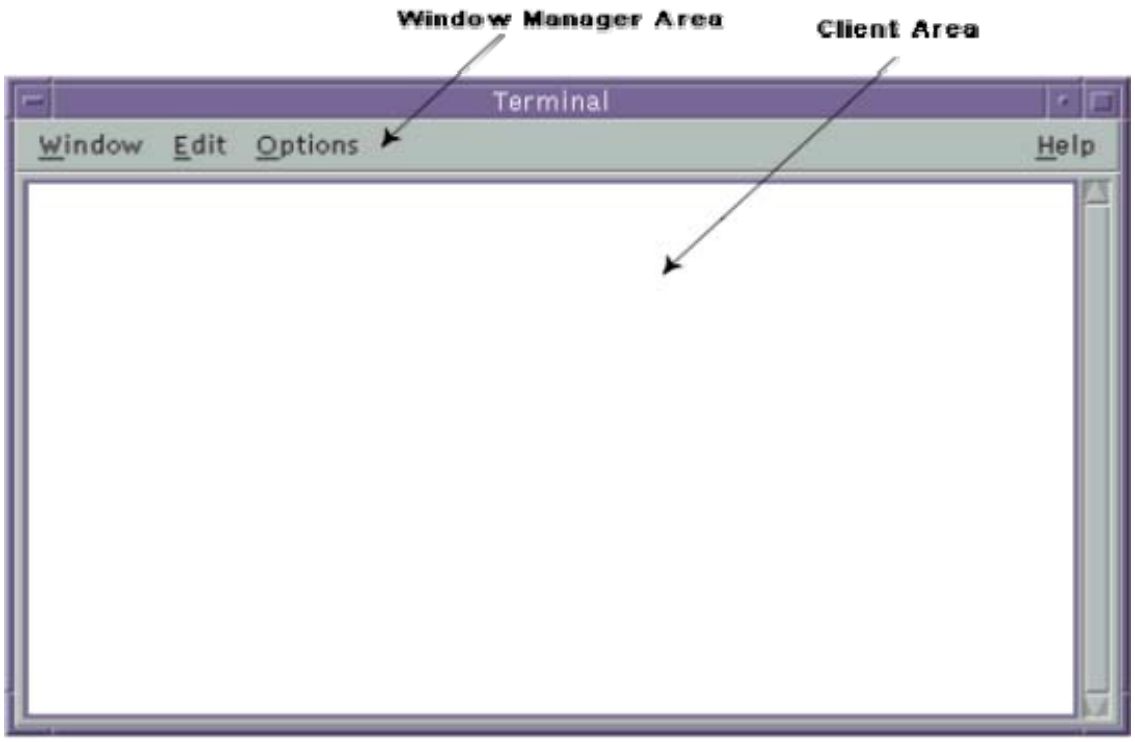

Fig. 9.6: Window with client and user areas marked

Window Management System

Window Management System (WMS)

A graphical user interface typically consists of a large number of windows. Therefore, it is necessary to have some systematic way to manage these windows. Most graphical user interface development environments do this through a window management system (WMS). A window management system is primarily a resource manager. It keeps track of the screen area resource and allocates it to the different windows that seek to use the screen. From a broader perspective, a WMS can be considered as a user interface management system (UIMS) – which not only does resource management, but also provides the basic behavior to the windows and provides several utility routines to the application programmer for user interface development. A WMS simplifies the task of a GUI designer to a great extent by providing the basic behavior to the various windows such as move, resize, iconify, etc. as soon as they are created and by providing the basic routines to manipulate the windows from the application such as creating, destroying, changing different attributes of the windows, and drawing text, lines, etc.

A WMS consists of two parts (as shown in fig. 9.7):

- a window manager, and
- a window system.

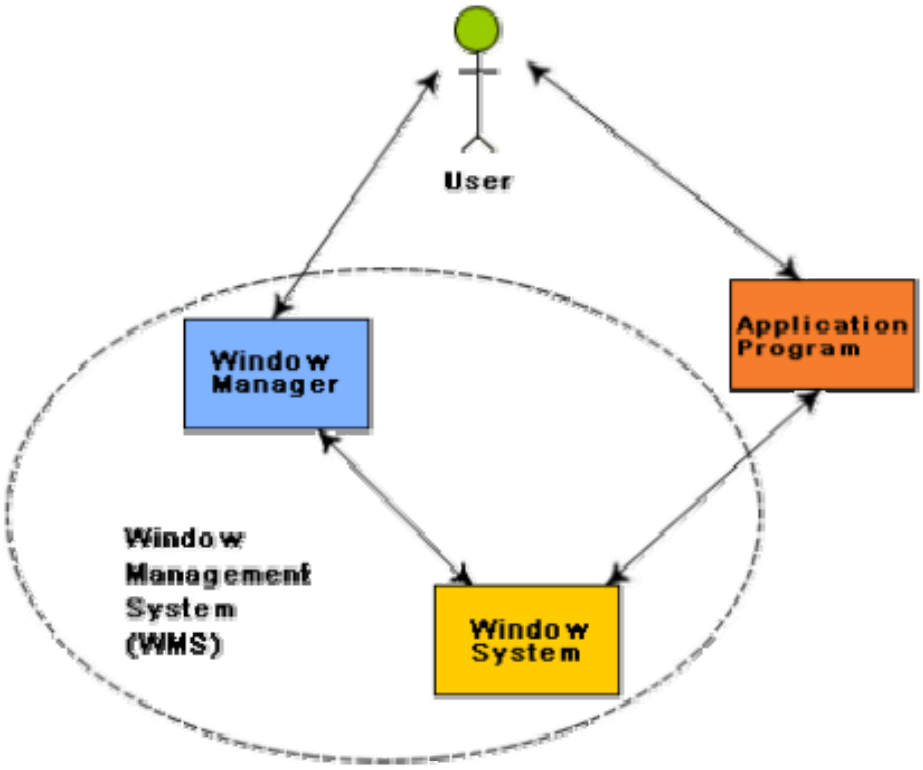

Fig. 9.7: Window Management System

Window Manager and Window System

Window manager is the component of WMS with which the end user interacts to do various window-related operations such as window repositioning, window resizing, iconification, etc. The window manager is built on the top of the window system in the sense that it makes use of various services provided by the window system. The window manager and not the window system determines how the windows look and behave. In fact, several kinds of window managers can be developed based on the same window system. The window manager can be considered as a special kind of client that makes use of the services (function calls) supported by the window system. The application programmer can also directly invoke the services of the window system to develop the user interface. The relationship between the window manager, window system, and the application program is shown in fig. 9.7. This figure shows that the end-user can either interact with the application itself or with the window manager (resize, move, etc.) and both

the application and the window manager invoke services of the window manager.

Window Manager

The window manager is responsible for managing and maintaining the non-client area of a window. Window manager manages the real-estate policy, provides look and feel of each individual window.

Types of widgets (window objects)

Different interface programming packages support different widget sets. However, a surprising number of them contain similar kinds of widgets, so that one can think of a generic widget set which is applicable to most interfaces. The following widgets are representatives of this generic class.

Label widget. This is probably one of the simplest widgets. A label widget does nothing except to display a label, i.e. it does not have any other interaction capabilities and is not sensitive to mouse clicks. A label widget is often used as a part of other widgets.

Container widget. These widgets do not stand by themselves, but exist merely to contain other widgets. Other widgets are created as children of the container widget. When the container widget is moved or resized, its children widget also get moved or resized. A container widget has no callback routines associated with it.

Pop-up menu. These are transient and task specific. A pop-up menu appears upon pressing the mouse button, irrespective of the mouse position.

Pull-down menu. These are more permanent and general. You have to move the cursor to a specific location and pull down this type of menu.

Dialog boxes. We often need to select multiple elements from a selection list. A dialog box remains visible until explicitly dismissed by the user. A dialog box can include areas for entering text as well as values. If an apply command is supported in a dialog box, the newly entered values can be tried without dismissing the box. Through most dialog boxes ask you to enter some information, there are some dialog boxes which are merely informative, alerting you to a problem with your system or an error you have made. Generally, these boxes ask you to read the information presented and then click *OK* to dismiss the box.

Push button. A push button contains key words or pictures that describe the action that is triggered when you activate the button. Usually, the

action related to a push button occurs immediately when you click a push button unless it contains an ellipsis (…). A push button with an ellipsis generally indicates that another dialog box will appear.

Radio buttons. A set of radio buttons is used when only one option has to be selected out of many options. A radio button is a hollow circle followed by text describing the option it stands for. When a radio button is selected, it appears filled and the previously selected radio button from the group is unselected. Only one radio button from a group can be selected at any time. This operation is similar to that of the band selection buttons that were available in old radios.

Combo boxes. A combo box looks like a button until the user interacts with it. When the user presses or clicks it, the combo box displays a menu of items to choose from. Normally a combo box is used to display either one-of-many choices when space is limited, the number of choices is large, or when the menu items are computed at run-time.

X-Window.

The X-window functions are low-level functions written in C language which can be called from application programs. But only the very serious application designer would program directly using the X-windows library routines. Built on top of X-windows are higher-level functions collectively called Xtoolkit. Xtoolkit consists of a set of basic widgets and a set of routines to manipulate these widgets. One of the most widely used widget sets is X/Motif. Digital Equipment Corporation (DEC) used the basic X-window functions to develop its own look and feel for interface designs called DECWindows.

Popularity of X-Window

One of the important reasons behind the extreme popularity of the X-window system is probably due to the fact that it allows development of portable GUIs. Applications developed using X-window system are device-independent. Also, applications developed using the X-window system become network independent in the sense that the interface would work just as well on a terminal connected anywhere on the same network as the computer running the application is. Network-independent GUI operation has been schematically represented in the fig. 9.8. Here, “A” is the computer application in which the application is running. “B” can be any computer on the network from where interaction with the application can be made. Network-independent GUI was pioneered by the X-window system in the mid-eighties at MIT (Massachusetts Institute of Technology) with support from DEC (Digital Equipment Corporation). Now-a-days many user interface development systems support network-independent GUI development, e.g. the AWT and Swing components of Java.

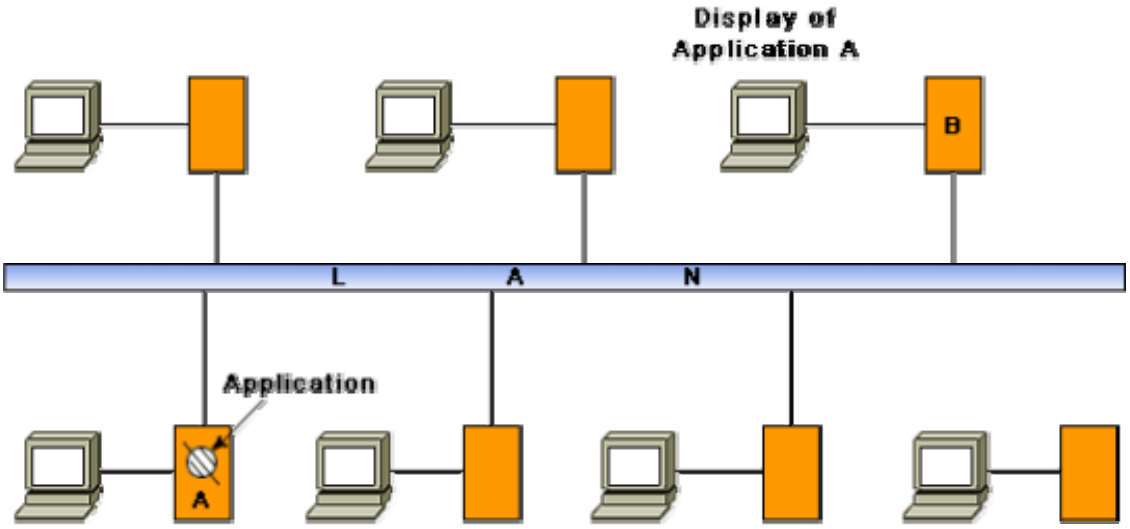

Fig. 9.8: Network-independent GUI

Architecture of an X-System

The X-architecture is pictorially depicted in fig. 9.9. The different terms used in this diagram are explained below.

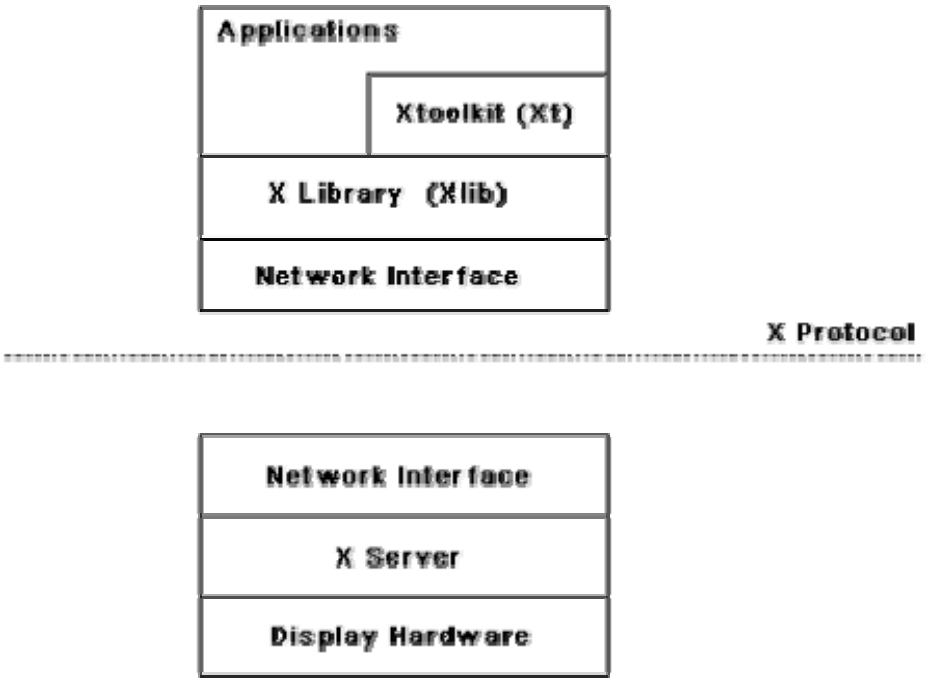

Fig. 9.9: Architecture of the X-System

X-server. The X server runs on the hardware to which the display and keyboard attached. The X server performs low-level graphics, manages window, and user input functions. The X server controls accesses to a bit-mapped graphics display resource and manages it.

X-protocol. The X protocol defines the format of the requests between client applications and display servers over the network. The X protocol is designed to be independent of hardware, operating systems, underlying network protocol, and the programming language used.

X-library (Xlib). The Xlib provides a set of about 300 utility routines for applications to call. These routines convert procedure calls into requests that are transmitted to the server. Xlib provides low level primitives for developing an user interface, such as displaying a window, drawing characteristics and graphics on the window, waiting for specific events, etc.

Xtoolkit (Xt). The Xtoolkit consists of two parts: the intrinsics and the widgets. We have already seen that widgets are predefined user interface components such as scroll bars, push buttons, etc. for designing GUIs. Intrinsics are a set of about a dozen library routines that allow a programmer to combine a set of widgets into a user interface. In order to develop a user interface, the designer has to put together the set of components (widgets) he needs, and then he needs to define the characteristics (called resources) and behavior of these widgets by using the intrinsic routines to complete the development of the interface. Therefore, developing an interface using Xtoolkit is much easier than developing the same interface using only X library.

Visual Programming

Visual programming is the drag and drop style of program development. In this style of user interface development, a number of visual objects (icons) representing the GUI components are provided by the programming environment. The application programmer can easily develop the user interface by dragging the required component types (e.g. menu, forms, etc.) from the displayed icons and placing them wherever required. Thus, visual programming can be considered as program development through manipulation of several visual objects. Reuse of program components in the form of visual objects is an important aspect of this style of programming. Though popular for user interface development, this style of programming can be used for other applications such as Computer-Aided Design application (e.g. factory design), simulation, etc. User

interface development using a visual programming language greatly reduces the effort required to develop the interface.

Examples of popular visual programming languages are Visual Basic, Visual C++, etc. Visual C++ provides tools for building programs with window-based user interfaces for Microsoft Windows environments. In Visual C++, menu bars, icons, and dialog boxes, etc. can be designed easily before adding them to program. These objects are called as resources. Shape, location, type, and size of the dialog boxes can be designed before writing any C++ code for the application.

Difference between user-centered design and design by users

- User-centered design is the theme of almost all modern user interface design techniques. However, user-centered design does not mean design by users. One should not get the users to design the interface, nor should one assume that the user's opinion of which design alternative is superior is always right.
- Users have good knowledge of the tasks they have to perform, they also know whether they find an interface easy to learn and use but they have less understanding and experience in GUI design than the GUI developers.

Implications of human cognition capabilities on user interface design

An area of human-computer interaction where extensive research has been conducted is how human cognitive capabilities and limitations influence the way an interface should be designed. The following are some of the prominent issues extensively discussed in the literature.

- **Limited memory.** Humans can remember at most seven unrelated items of information for short periods of time. Therefore, the GUI designer should not require the user to remember too many items of information at a time. It is the GUI designer's responsibility to anticipate what information the user will need at what point of each task and to ensure that the relevant information is displayed for the user to see. Showing the user some information at some point, and then asking him to recollect that information in a different screen where they no longer see the information places a memory burden on the user and should be avoided wherever possible.

- **Frequent task closure.** Doing a task (except for very trivial tasks) requires doing several subtasks. When the system gives a clear feedback

to the user that a task has been successfully completed, the user gets a sense of achievement and relief. The user can clear out information regarding the completed task from memory. This is known as task closure. When the overall task is fairly big and complex, it should be divided into subtasks, each of which has a clear subgoal which can be a closure point.

- **Recognition rather than recall.** Information recall incurs a larger memory burden on the users and is to be avoided as far as possible. On the other hand, recognition of information from the alternatives shown to him is more acceptable.

- **Procedural versus object-oriented.** Procedural designs focus on tasks, prompting the user in each step of the task, giving them few options for anything else. This approach is the best applied in situations where the tasks are narrow and well-defined or where the users are inexperienced, such as an ATM. An object-oriented interface on the other hand focuses on objects. This allows the users a wide range of options.

GUI design methodology

GUI design methodology consists of the following important steps:

- Examine the use case model of the software. Interview, discuss, and review the GUI issues with the end-users.
- Task and object modeling
- Metaphor selection
- Interaction design and rough layout
- Detailed presentation and graphics design
- GUI construction
- Usability evaluation

The starting point for GUI design is the use case model. This captures the important tasks the users need to perform using the software. As far as possible, a user interface should be developed using one or more metaphors. Metaphors help in interface development at lower effort and reduced costs for training the users. Over time, people have developed efficient methods of dealing with some commonly occurring situations. These solutions are the themes of the metaphors. Metaphors can also be based on physical objects such as a visitor's book, a catalog, a pen, a brush, a scissor, etc. A solution based on metaphors is easily understood by the users, reducing learning time and training costs. Some commonly used metaphors are the following:

- White board
- Shopping cart
- Desktop
- Editor's work bench
- White page
- Yellow page
- Office cabinet
- Post box
- Bulletin board
- Visitor's book

Task and Object Modeling

A task is a human activity intended to achieve some goals. Example of task goals can be:

- reserve an airline seat
- buy an item
- transfer money from one account to another
- book a cargo for transmission to an address

A task model is an abstract model of the structure of a task. A task model should show the structure of the subtasks that the user needs to perform to achieve the overall task goal. Each task can be modeled as a hierarchy of subtasks. A task model can be drawn using a graphical notation similar to the activity network model. Tasks can be drawn as boxes with lines showing how a task is broken down into subtasks. An underlined task box would mean that no further decomposition of the task is required. An example decomposition of a task into subtasks is shown in fig. 9.10.

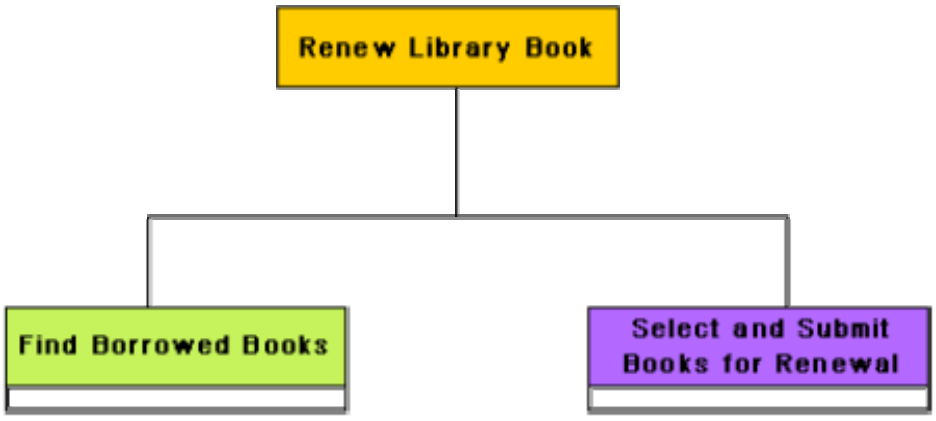

Fig. 9.10: Decomposition of a task into subtasks

Selecting a metaphor

The first place one should look for while trying to identify the candidate metaphors is the set of parallels to objects, tasks, and terminologies of the use cases. If no obvious metaphors can be found, then the designer can fall back on the metaphors of the physical world of concrete objects. The appropriateness of each candidate metaphor should be tested by restating the objects and tasks of the user interface model in terms of the metaphor. Another criterion that can be used to judge metaphors is that the metaphor should be as simple as possible, the operations using the metaphor should be clear and coherent and it should fit with the users' 'common sense' knowledge. For example, it would indeed be very awkward and a nuisance for the users if the scissor metaphor is used to glue different items.

Example: We need to develop the interface for the automation shop, where the users can examine the contents of the shop through a web interface and can order them.

Several metaphors are possible for different parts of this problem.

- Different items can be picked up from **racks** and examined. The user can request for the **catalog** associated with the items by clicking on the item.
- Related items can be picked from the **drawers** of an item cabinet.
- The items can be organized in the form of a book, similar to the way information about electronic components are organized in a semiconductor hand book.

Once the users make up their mind about an item they wish to buy, they can put them into a **shopping cart**.

User interface inspection

Nielson [Niel94] studied common usability problems and built a check list of points which can be easily checked for an interface. The following check list is based on the work of Nielson [Niel94].

Visibility of the system status. The system should as far as possible keep the user informed about the status of the system and what is going on.

Match between the system and the real world. The system should speak the user's language words, phrases, and concepts familiar to that used by the user, rather than using system-oriented terms.

Undoing mistakes. The user should feel that he is in control rather than feeling helpless or to be at the control of the system. An important step toward this is that the users should be able to undo and redo operations.

Consistency. The user should not have to wonder whether different words, concepts, and operations mean the same thing in different situations.

Recognition rather than recall. The user should not have to recall information which was presented in another screen. All data and instructions should be visible on the screen for selection by the user.

Support for multiple skill levels. Provision of accelerations for experienced users allows them to efficiently carry out the actions they frequently require to perform.

Aesthetic and minimalist design. Dialogs should not contain information which are irrelevant and are rarely needed. Every extra unit of information in a dialog competes with the relevant units and diminishes their visibility.

Help and error messages. These should be expressed in plain language (no codes), precisely indicating the problem, and constructively suggesting a solution.

Error prevention. Error possibilities should be minimized. A key principle in this regard is to prevent the user from entering wrong values. In situations where a choice has to be made from among a discrete set of values, the control should present only the valid values using a drop-down list, a set of option buttons or a similar multichoice control. When a specific format is required for attribute data, the entered data should be validated when the user attempts to submit the data.

The following questions have been designed to test the identified objectives for this module:

1. List five desirable characteristics that a good user interface should possess.

2. What is the difference between user guidance and on-line help system in the user interface of a software system?

3. Discuss the different ways in which on-line help can be provided to a user while he is executing the software.

4. What is the difference between a mode-based interface and a modeless interface?

5. Compare the relative advantages of textual and graphical user interfaces.

6. Compare the relative advantages of command language, menu-based, and direct manipulation interfaces.

7. List the important advantages and disadvantages of a command language interface.

8. List the important advantages and disadvantages of a menu-based interface.

9. Compare the relative advantages of scrolling menu, hierarchical menu, and walking menu as techniques for organizing user commands.

10. What do you understand by an iconic interface? Explain how you can issue commands using an iconic interface.

11. List the important advantages and disadvantages of a direct manipulation interface.

12. Suppose you have been asked to design the user interface of a large software product. Would you choose a menu-based, a direct manipulation, a command language-based, or a mixture of all these types of interfaces to develop the interface for your product? Justify your choice.

13. Explain the reason of popularity of component-based GUI development.

14. What is Window Management System (WMS)? Represent the main components of a WMS in a schematic diagram and explain their roles.

15. What are the advantages of using a Window Management System (WMS) for a GUI design? Name some commercially available Window Management Systems.

16. Explain the responsibilities of a window manager in the context of Window Management System (WMS).

17. Discuss the architecture of the X window system.

18. What are the important advantages of using the X window system for developing graphical user interfaces?

19. What do you understand by visual programming?

20. Distinguish between a user-centric interface design and interface design by users.

21. How does the human cognition capabilities and limitations influence human-computer user interface designing?

22. What do you understand by a metaphor in a user interface design? Is a metaphor-based user interface design advantageous? Justify it.

23. List a few metaphors which can be used for user interface design.

24. What is meant by a task model? Explain it with examples.

25. Do prepare a check list for user interface inspection.

Mark all options which are true.

1. The interface of a software product which is supporting a large number of commands can be developed using
   - □ mode-based interface
   - □ modeless interface
   - □ either mode-based interface or modeless interface
   - □ neither mode-based interface nor modeless interface

2. The interface that can be implemented even on cheap alphanumeric terminals is
   - □ menu-based interface
   - □ command language-based interface
   - □ iconic interface
   - □ none of them

3. The term “iconic interface” is applicable to
   □ command language-based interface
   □ menu-based interface
   □ direct manipulation interface
   □ none of the above

4. A command composition facility can be made available in case of
   □ menu-based interface
   □ command language-based interface
   □ direct manipulation interface
   □ none of the above

5. A development style based on widgets is called
   □ command language-based GUI development style
   □ component-based GUI development style
   □ menu-based GUI development style
   □ direct manipulation based GUI development style

6. In the case of X Window architecture, the format of the requests between client applications and display servers over the network is defined by
   □ X-server
   □ X-library
   □ X-protocol
   □ Xtoolkit

7. The low level primitives for developing a user interface, such as displaying a window, drawing characteristics and graphics on the window etc. in case of X Window system are provided by

   □ X-server
   □ X-library
   □ X-protocol
   □ Xtoolkit

Mark the following as either True or False. Justify your answer.

1. A good user interface must provide feedback to various user actions.

2. A good user interface should support multiple levels of sophistication of command issue procedure for different categories of users.

3. In the context of user interface, “Guidance” and “On-line Help” are used for the same purpose.

4. Novice users normally prefer command language interfaces over both menu-based and iconic interfaces.

5. Intrinsics of Xtoolkit of a X Window system are a set of about dozen library routines that allow a programmer/user interface designer to combine a set of widgets into a user interface.

6. Visual programming style is restricted to user interface development only.

7. Visual programming can be considered as program development through manipulation of several visual objects.

8. For modern user interfaces, LOC is an accurate measure of the size of the interface.

9. When a window is a modal dialog, no other windows in the application are accessible until the current window is closed.

10. Graphical User Interfaces should let the user recall commands rather than have him recognize commands from a repertoire of displayed commands.

11. In case of good user interface design, the GUI designer should try to reduce the number of screens by cramping as much information on a screen as possible.