Unit 11 Software Tools

Structure:

11.1 Introduction

In the previous unit, you have studied about characteristics of text and messages for user interaction. You also learnt the use of icons, multimedia, colors in user interface design.

In this unit, you will study about specification methods for user interaction and user interface software tools.

Almost as long as there have been user interfaces, there have been special software systems and tools to help design and implement the user interface software. Many of these tools have demonstrated significant productivity gains for programmers, and have become important commercial products. Others have proven less successful at supporting the kinds of user interfaces people want to build.

Objectives

After studying this unit, you should be able to:

- explain various specification methods in the design of user interface
- discuss the features of interface building tools and importance of user interface tools

11.2 Specification Methods

As we all know that Computers today are used for a broad range of applications. User interface design guidelines cannot be applied usefully in every case. Some computers may be embedded as components in larger systems, so they communicate only with other computers and not directly with human users. When there is no user interface, then no user interface design guidelines are needed. To the extent that information systems support human users performing defined tasks, careful design of the user-system interface will be needed to ensure effective system operation. The guidelines proposed in this unit are intended to improve user interface design for such information systems.

Users of information systems interact with a computer in order to accomplish information handling tasks necessary to get their jobs done. They differ in ability, training and job experience. They may be keenly concerned with task performance, but may have little knowledge of (or interest in) the computers themselves. Design of the user-system interface must take account of the following human factors.

1. Design requires a good notation to record and discuss alternate possibilities: The default language for specifications in any field is natural language, e.g., English Communication medium, e.g., sketchpad, or blackboard.
2. Natural language specifications tend to be: lengthy, vague & ambiguous. Therefore often are difficult to prove: correct, consistent and complete. Backus-Naur Form (a.k.a. Backus Normal Form or BNF).
3. High-level components are described as non terminal and specific strings are described as terminals

Now let us discuss on User interaction specification, most important approaches/notations to specifying interaction are State-Transition Diagrams (STD), Petri Nets (PN), Goal-Operation-Methods-Selections (GOMS) and User Action Notation (UAN). All these approaches aim to provide more detailed descriptions of interaction between user and system. Let us discuss these approaches in detail

State Transition Diagram (STD)

State Transition Diagram (STD) is a set of nodes that represents system states and a set of links between the nodes that represents possible transitions. The basic elements of these are: state and state transition where State is a set of values that describe an object (its condition/situation) at a specific moment in time (state is determined based on the attribute values) and State transition is a relationship indicating a state change. Fig 11.1 shows an example for state transition diagram to draw circle.

***STD example**:*

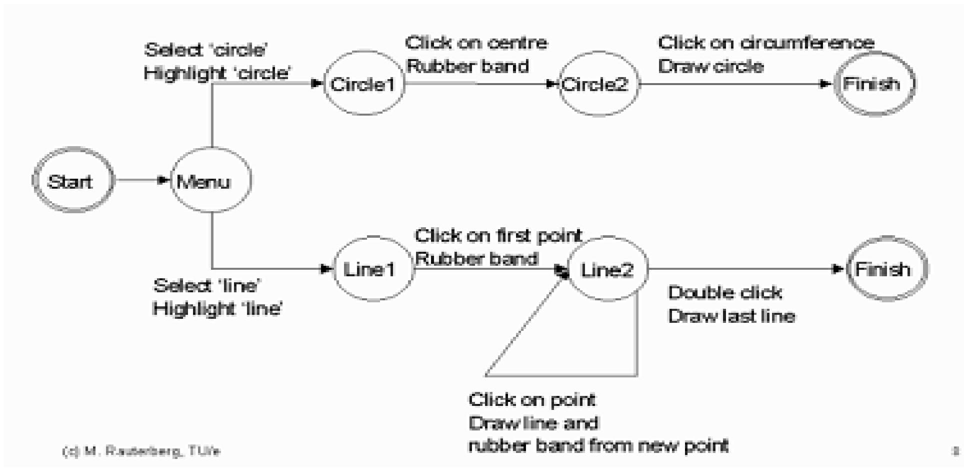


Fig. 11.1: STD example: Draw circle

Petri Nets (PN)

Petri Nets (PN) was first introduced by Carl Adam Petri in 1962. It is a diagrammatic tool to model concurrency and synchronization in distributed systems. Very similar to state transition diagrams petri nuts areused as a visual communication aid to model the system behavior and it is based on strong mathematical foundation.

The Basic elements of these approaches are illustrated in the figure 11.2.

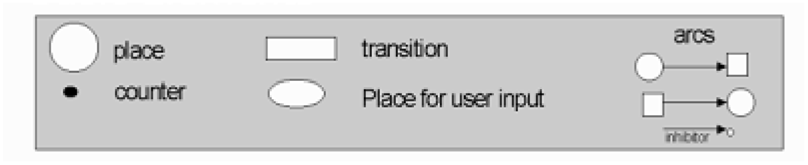


Fig. 11.2: Petri Nets

PN consists of three types of components: Places (circles), Transitions (rectangles) and Arcs (arrows). Places represent possible states of the system, Transitions are events or actions which cause the change of state, and every arc simply connects a place with a transition or a transition with a place.

Formal definition for PN is given below

A Petri Net is a 5 tuple

PN (P, T, IN, OUT, M)

Where:

$P=\{p_1,p_2, , , , , , ,p_n\}$ is a finite set of places

$T=\{t_1,t_2, , , , , , , ,t_n)$ is a finite set of transitions

$IN= (PxT)\rightarrow S$

$OUT= (TxP)\rightarrow S$

M=Marking vector

IN are input functions defining directed arcs from places to transitions

OUT are output functions defining directed arcs from transitions to places

S is a set of all nonnegative integers K such that:

- If K=1, a directed arc is drawn without a label
- If K>1, a directed arc is drawn with a label K
- If K=0, no arc is drawn

There are some rules for PN Transitions, these rules are called as firingrules, and they are

- A specific transition t_i is said to be enabled, if each input place p_i is marked with at least $w(p_i, t_i)$ tokens where, $w(p_i, t_i)$ is the weight of the arc from p_i to t_i
- An enabled transition may or may not fire depending on whether or not the event actually takes place
- The firing of an enabled transition t_i removes $w(p_i, t_i)$ tokens from each input place p_i of t_i and adds $w(p_j, t_i)$ tokens to each output place p_j of $t_{i.}$

Where,

$w(p_i, t_i)$ is the weight of the arc from input place p_i to t_i and

$w(p_j, t_i)$ is the weight of the arc from t_i to output place p_j

In PN change of states is denoted by a movement of token(s) (black dots) from place(s) to place(s) which is caused by the firing of a transition. The firing represents the occurrence of an event or the action taken. The firing is subjected to the input conditions denoted by, token availability. A transition is friable or enabled when there are sufficient tokens in its input places. After firing, tokens will be transferred from the input places (old state) to the output places, denoting the new state.

Goal-Operation-Methods-Selections (GOMS)

GOMS is a family of user interface modeling techniques. Goals, Operators, Methods and Selection rules are Input for this approach. This approach is used various qualitative and quantitative measures, which are usefully approximations.

Members of GOMS family are Keystroke Level Model (KLM), Natural GOMS language (NGOMSL) and Critical Path Method (CPM) or Cognitive, Perceptual and Motor GOMS (CPM-GOMS). NGOMSL and CPM-GOMS are Other GOMS techniques.

GOMS can model the task which are goal directed.GOMS can also model task which have a routine cognitive skill. GOMS model both Serial and parallel tasks.

GOMS is applied to compare user interface designs, profiling, for sensitivity and parametric analysis and for building a help system where GOMS modeling makes user tasks and goals explicit and can suggest questions, users will ask and the answers.

Advantages of GOMS:

- Gives qualitative and quantitative measures
- Model explains the results
- Less work than user study-no users!
- Easy to modify when UI is revised
- Research: tools to aid modeling process, since it can still be tedious

Disadvantages of GOMS:

- Not as easy as HE, guidelines etc
- Takes lots of time, skill and effort
- Only works for goal-directed tasks
- Assumes tasks performed by experts without error
- Does not address several UI issues:
  - Readability, memorizability of icons, commands

User Action Notation (UAN)

User Action Notation (UAN) is developed by Hix and Hartson. Further this approach was refined by Hartson and Gray. UAN is neutral about the user and interface technology. It aims to show how tasks match computer devices. The elements of UAN are symbols & operators, conditions &

options, tables of user action, feedback & system action/state and temporal relations & constraints.

Now let us know the UAN symbols and operators, existing UAN uses special characters for mouse movements and button actions

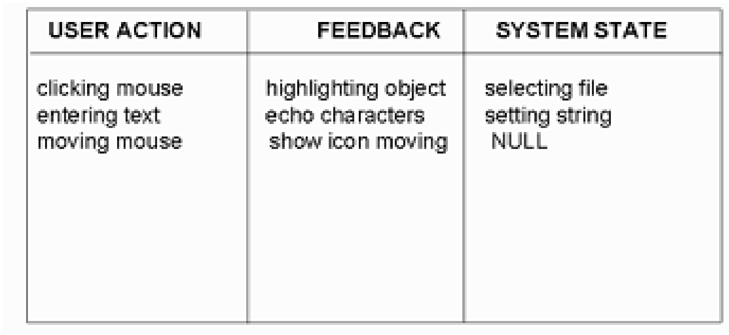
Move_mouse (x, y)*
Release button (x', y')
Highlight (icon)
De_highlight (icon)
File = select ( )

UAN uses the following conditions and options

- While (condition) TASK
- If (condition) THEN task
- Iteration A* or A+
- Waiting can be an operation on a task

The UAN Table 11.1 provides information on user action, feedback and system state.

Table 11.1: UAN-three columned table



USER ACTION	FEEDBACK	SYSTEM STATE
clicking mouse	highlighting object	selecting file
entering text	echo characters	setting string
moving mouse	show icon moving	NULL

UAN: temporal relations

- Strict sequence [A, B]
  - B follows completion of A
- Order independence [A & B]
  - A & B can be done in any order
- Concurrence with [A ıı B]
  - A & B are done simultaneously
- Interruptible by [A -> B]
  - A can interrupt B

- Interleavable [A < | > B]
  - Swapping between A & B

Self Assessment Questions

1. The three components of Petri Nets are ____________, ________ & ___________.
2. Members of GOMS family are ___________, __________ and ____________.
3. Who developed UAN?
   ____

11.3 Interface-Building Tools

What is the user-system interface? In common usage, the phrase is broadly defined to include all aspects of system design that affect system use. This section, however, is concerned more narrowly with the user interface to computer-based information systems, i.e., with those aspects of system design that influence a user's participation in information handling tasks.

This section focuses even more narrowly on those design features of the user interface that are implemented via software (i.e., the design of computer program logic) rather than hardware (the design of equipment). The guidelines proposed here are generally worded in terms of the functions that a user must perform, and the functional capabilities that a designer should provide, rather than the particular physical devices that might be used to implement those functions. Thus a particular guideline might deal with "pointing" as a function, with no necessary recommendation whether pointing should be accomplished via touch display or lightpen or any other physical device.

The user interface of a computer program is the part that handles the output to the display and the input from the person using the program. The rest of the program is called the application or the application semantics.

User Interface Software Tools: user interface software toolshelp to provide the user and developer with a realistic impression of how the final product may look like function (e.g., a prototype for a menu system) such prototypes are naturally limited (e.g., restricted to only 1 or 2 tasks) Examples: Macromedia Director, Flash, Visual Basic, JBuilder

Four different classes of people are involved with user interface software, and it is important to have different names for them to avoid confusion. The first is the person using the resulting program, who is called the end-user orjust user. The next person creates the user interface of the program, and is called the user interface designer or just designer. Working with the user interface designer will be the person who writes the software for the rest ofthe application. This person is called the application programmer. The designer may use special user interface tools which are provided to help create user interfaces. These tools are created by the tool creator.

Note that the designer will be a user of the software created by the tool creator, but westill do not use the term "user" here to avoid confusion with the end-user. Although this classification discusses each role as a different person, in fact, there may be many people in each role, or one person may perform multiple roles. The general term programmer is used for anyone who writes code, andmay be a designer, application programmer, or tool creator.

It is obvious that software is not the only significant factor influencing user performance. Other aspects of user interface design are clearly important, including workstation design, physical display characteristics, keyboard layout, environmental factors such as illumination and noise.

Some computers are designed as general tools which can be adapted by skilled users for whatever purpose they desire. The particular tasks for which a general-purpose computer might be used are not defined in advance by the designer. Instead, a user must provide exact instructions to program the computer to perform any task at hand. The designer may try to ensure that the computer can process appropriate programming languages, but otherwise is not concerned with explicit design of a user interface.

Other computer systems are designed to help particular users perform specific tasks. Such computer applications are referred to here as information systems. Applications of information systems range from relatively simple data entry and retrieval (e.g., airline reservations) through more complex monitoring and control tasks (inventory control, process control, air traffic control) to jobs requiring long-term analysis and planning. Military command, control and communication systems span that broad range of information system applications.

In this section, we survey different tools and techniques for prototyping user interfaces, ranging from paper and pencil to draw mockups of displays to sophisticated interface construction toolkits. We view prototyping as an information gathering process, so we will compare the tools and techniques according to two criteria. The first criterion is a measure of the completeness andvariety of the information that a tool or technique can help acquire, given that several different kinds of information are needed to design and build a good user interface. The second criterion is the ability of the tools to expedite the information gathering process in order to minimize the cost of the prototyping process and to maximize its effectiveness.

11.3.1 Products of the Prototyping Process

The main purpose of a prototype is to allow developers to acquire the information needed to successfully build a system. We first review the kinds of information needed to develop asuccessful interface, and then we will compare prototyping tools and techniques in terms of their ability to maximize the effectiveness and minimize the cost of acquiring the information.

To build successful interface developers we need to acquire several kinds of information about the system to be built. The required information falls into the following categories:

Task specification: The task specification is needed for developers to understand what services to provide in a system and how to deliver them to the end-users in order to help them to perform their tasks more effectively. Prototypes help developers better understand the tasks that users need toper form by letting developers see users in action with the system, and get feedback about the effectiveness of a system. In addition, successful systems often change the nature of the tasks that users perform, enabling them to do tasks they could not accomplish before. Prototypes let developers see how a system might change the tasks that users perform, and allow them to designa better system.

System functionality: This information specifies the requirements of the software modules that the interface software calls upon to fetch data display, and to modify data in response to user requests. Different interface designs often impose different requirements on system functionality. They allow interface designers to explore design alternatives without having to wait for programmers to revise thesystem functionality.

Interface functionality: This specification captures the"content" of the interface, abstracting away from "style" details such as font, color, etc. It is important for developers to understand the interface at this abstract level. For example, before worrying about style issues of a display, developers should understand whether the displaypresents the right amount of data at the right time to help users perform their tasks.

Screen layouts and behavior: This information is of primary importance because it defines what users can see and do. Prototyping is very useful to acquire this information.

Design rationale: This information is useful for many reasons. It can be used to achieve consistency in the interface, to guide extensions to the interface of a previous version of the system, to review and justify designs with management, etc.

Response times: Prototypes let developers see the users in action and understand the required response time’s for effective system usage.

Reusable code: Building a prototype can be expensive, and the cost is harder to justify when the prototype's implementation cannot be reused in the implementation of the realsystem.

11.3.2 User interface mockup tools

The essence of user interface prototyping is to construct a small scale version of an interactive system to collect information to guide its construction. Interface prototypes can be built with a large variety of tools, ranging from paper and pencil to draw mockups of displays to sophisticated interface construction toolkits. In this section we describe different categories of tools that can be used to prototype interfaces, highlighting their special strengths and weaknesses. The following are some of the tools

Paper and pencil:

Paper and pencil are perhaps the most popular tools, one uses to describe interface designs toothers. Under this category we also include electronic versions of these tools such as drawing, painting and text editors. Paper and pencil are prototyping tools with much strength. They are easy to use most people can draw boxes with buttons, menus and scribbles representing the objects in an application domain. Paper and pencil allows extensive control over details of the design. Control is not even limited by our ability to draw, because we can always draw a scribble and tell others what it means. Paper and pencil are alsovery useful for capturing different kinds of information.

The main weaknesses of the paper and pencil technique are it is very awkward to capture behavior, and that the interface prototypes are not executable.

Facade tools:
Facade tools are essentially drawing editors with an ability to specify input behavior. We call them facade tools because they allow developers to construct screens that look and behave likethe screens of the real application, except that there is no "application" behind them. The screensdisplay canned data, and the behaviors either switch to another canned screen, or update thescreen with a new set of scanned data.

Astound is a presentation preparation package. It allows users to produce a sequence of slides containing both text and graphics. Users can also add to a slide buttons with associated behaviors such as jumping to another slide, or making elements of the slide appear and disappear. Astound provides sophisticated animation capabilities to dramatize the transitions between slides. Astound can be used as an interface prototyping tool. Developers can draw the displays of anapplication and use the buttons to show the sequence of displays that users will need to traverse to accomplish different tasks.

HyperCard is a tool to build hypertext applications. Hypercard applications are built with two kinds of abstractions: cards and stacks of cards. Cards can contain a variety of fields and pictures.The fields can be type-in areas, buttons, menus and other WIMP interface building blocks (i.e.,interfaces consisting of Windows, Icons, Menus and Pointing).

HyperCard has three modes of operation: end-user, simple authoring and application developer.

- The end-user mode allows theuser to interact with the HyperCard application, but not to extend it in any way.
- The simple authoring mode allows the user to insert links between cards, change layouts and edit scripts.

- The application developer mode gives developers full access to all HyperCard capabilities, including the ability to define new cards and new fields, and to define and modify scripts.
- HyperCard's simple authoring and developer modes provide excellent facilities for prototyping interfaces. The simple authoring mode allows all members of the design team to be involved inscreen design and simple behavior definition. End-user mode allows users and developers to test the interface.

Strengths of HyperCard:

- Facade tools retain most of the benefits of paper and pencil, while adding the ability to describe behavior and to execute the prototypes.
- Many facade tools can give end-users the illusion of interacting with the real application, so they can be used to get very reliable feedback from end-users.

Weaknesses of HyperCard:

- The main weakness of facade tools appears to be that they do not produce reusable code that canbe used to build the real application, so the implementation effort in building the prototype is lost.
- Additionally, since the prototyping and the implementation tools are so different, the prototype and the application might be built by different teams of people, and many of the lessons learned while building the prototype remain in the minds of the prototyping team, and do not carry over effectively to the implementation of the system.
- A potential problem of facade tools is that they can over-sell the capabilities of an application giving the illusion that a more sophisticated application will be constructed than what is feasible given budget, time and technology constraints.

Self Assessment Questions

4. ____________ is a tool to build hypertext applications.
5. HyperCard applications are built with two kinds of abstractions they are ____________ & ___________.
6. The three modes of operation of HyperCard are ____________, ____________ & __________________.
7. ______________ is a presentation preparation package.

Model-based tools:

Model-based interface development is a new paradigm for developing interfaces. The model-based paradigm uses a central database to store adescription of all aspects of an interface design. This central description is called a **model,** and typically contains information about the tasks that users are expected to performusing the application, the data of the application, the commands that users can perform the presentation and behavior of the interface, and the characteristics of users.

A standard software module called an **interface generator**, or **runtime system** uses the model as input, and maps the state of the application into the windows that appear on a user's screen. The runtime system also accepts inputs from the user and invokes the appropriate commands. Interfaces are developed by using specialized design-time tools to build and refine models. Developers specify what features the interface should have, rather than write programs thatspecify how to make the computer exhibit the desired behavior.

The main advantage of the model-based approach over traditional interface development approaches is that it enables the construction and use of tools to provide assistance to interface developers making it possible to capture many of the products of the prototyping process such as task specifications, system functionality, interface functionality, screen layout and behavior, and several kinds of design rationale. This same knowledge is used by the run-time system to implement the interfaces.

Model-based tools have three main weaknesses:

- First, they are difficult to use compared to interface builders, facade tools and paper and pencil, even though current research is addressing these issues very actively.
- Second, they provide only moderate control over the details of highly graphical user interfaces, and
- Third, since the model-based technology is not mature, the tools are not sufficiently efficient and reliable for wide spread use.

Domain-specific tools:

Domain-specific tools are tools for building special kinds of applications (e.g., database applications) or applications with specific styles of interfaces (e.g., HyperCard). By focusing ona narrower domain, these tools can provide powerful facilities for constructing applications veryquickly. Here we look at these tools as prototyping tools because the effort for building applications is often so small that it compares to the effort needed to build prototypes using theother prototyping techniques discussed.

11.3.3 Importance of user interface tools

There are many advantages to using user interface software tools. These canbe classified into two main groups:

(1) The quality of the interfaces will be higher. This is because:

- Designs can be rapidly prototyped and implemented, possibly even before the application code is written.
- It is easier to incorporate changes discovered through user testing.
- There can be multiple user interfaces for the same application.
- More effort can be expended on the tool than may be practical on any single user interface since the tool will be used with many different applications.
- Different applications are more likely to have consistent user interfaces if they are created using the same user interface tool.
- It will be easier for a variety of specialists to be involved in designing the user interface, rather than having the user interface created entirely by programmers. Graphic artists, cognitive psychologists, and human factors specialists may all be involved. In particular, professional user interface designers, who may not be programmers, can be incharge of the overall design.

(2) The user interface code will be easier and more economical to create and maintain. This is because: Interface specifications can be represented, validated, and evaluated more easily. There will be less code to write; because much is supplied by the tools.There will be better modularization due to the separation of the user interface component from the application. This should allow the user interface to change without affecting the application, and a large class of changes to the application (such as changing the internal algorithms)should be possible without affecting the user interface.The level of programming expertise of the interface designers and implementers can be lower, because the tools hide much of the complexities of the underlying system. The reliability of the user interface will be higher, since the code for the user

interface is created automatically from a higher-level specification. It will be easier to port an application to different hardware and software environments since the device dependencies are isolated in the user interface tool.

11.3.4 Windowing Systems

A windowing system is a software package that helps the user monitor and control different contexts by separating them physically onto different partsof one or more display screens. Although most of today's systems provide tool kits on top of the windowing systems, as will be explained below, generally toolkits address only the drawing of widgets such as buttons, menus, and scroll bars. Thus, when the programmer wants to draw application-specificparts of the interface and allow the user to manipulate these, the window system interface must be used directly. Therefore, the windowing system'sprogramming interface has significant impact on most user interface programmers.

Structure of windowing systems

The window system, or base layer, implements the basic functionality of the windowing system. The two parts of this layer handle the display of graphics in windows (the output model) and the access to the various input devices (the input model), which usually includes a keyboard and a pointing device such as a mouse. The primary interface of the base layer is procedural, and is called the windowing system's application or program interface. The other layer of windowing system is the window manager or user interface. This includes all aspects that are visible to the user. The two parts of the user interface layer are the presentation, which is comprised of the pictures that the window manager displays, and the commands, which are how the user manipulates the windows and their contents.

Base layer

The base layer is the procedural interface to the windowing system. In the1970s and early 1980s, there were a large number of different windowing systems, each with a different procedural interface (at least one for each hardware platform). People writing software found this to be unacceptable because they wanted to be able to run their software on different platforms, but they would have to rewrite significant amounts of code to convert from one window system to another. The X windowing system was created to solve this problem by providing a hardware-independent interface to windows.

Output model

The output model is the set of procedures that an application can use to draw pictures on the screen. It is important that all output be directed through the window system so that the graphics primitives can be clipped to the window's borders. For example, if a program draws a line that would extend out of a window's borders, it must be clipped so that the contents of other, independent, windows are not overwritten. Most windowing systems provide special escapes that allow programs to draw directly to the screen, without using the window system's clipping. These operations can be much quicker, but are very dangerous and therefore should seldom be used. Most modern computers provide graphics hardware that is specially optimized to work efficiently with the window system.

Self Assessment Questions

8. _____________ tools are tools for building special kinds of applications.
9. ___________ is the procedural interface to the windowing system.

11.4 Summary

In this unit you learnt about Specification methods, Building tools for user interaction. The design of user interface software is not only expensive and time-consuming, but it is also critical for effective system performance. To be sure, users can sometimes compensate for poor design with extra effort. Probably no single user interface design flaw, in itself, will cause system failure. But there is a limit to how well users can adapt to a poorly designed interface. As one deficiency is added to another, the cumulative negative effects may eventually result in system failure, poor performance, and/or user complaints.

In a constrained environment, such as that of many military and commercial information systems, users may have little choice but to make do with whatever interface design is provided. There the symptoms of poor user interface design may appear in degraded performance. Frequent and/or serious errors in data handling may result from confusing user interface design. Tedious user procedures may slow data processing, resulting in longer queues at the checkout counter, the teller's window, the visa office, the truck dock, or any other workplace where the potential benefits of computer support are outweighed by an unintended increase in human effort.

In situations where degradation in system performance is not so easily measured, symptoms of poor user interface design may appear as user complaints. The system may be described as hard to learn, or clumsy, tiring and slow to use. The users' view of a system is conditioned chiefly by experience with its interface. If the user interface is unsatisfactory, the users' view of the system will be negative regardless of any niceties of internal computer processing.

In designing computer-based information systems, special attention must be given to software supporting the user interface. For the past several years, guidelines for designing user interface software have been compiled as a continuing effort sponsored by the Air Force Electronic Systems Division (ESD). This unit revises and proposes a comprehensive set of guidelines for design of user interface software in computer-based information systems

11.5 Terminal Questions

1. What is the use of user interface software tools?
2. What are the kinds of information required by the developers about the system to build a successful interface?
3. What is windowing system? Explain

11.6 Answers

Self Assessment Questions

1. Places, transitions & arcs
2. Keystroke Level Model (KLM), Natural GOMS language (NGOMSL) and Critical Path Method
3. Hix and Hartson
4. HyperCard
5. Cards & stacks of cards
6. End-user, simple authoring & application developer
7. Astound
8. Domain-specific
9. Base layer

Terminal Questions

1. User interface software tools help to provide the user and developer with a realistic impression of how the final product may look like function. (Refer section 11.3)
2. The kinds of information required by the developers about the system to build a successful interface are:
   - Task specification
   - System functionality
   - Interface functionality
   - Screen layouts and behavior
   - Design rationale
   - User feedback
   - Response times
   - Reusable code (Refer section 11.3.1)
3. A windowing system is a software package that helps the user monitor and control different contexts by separating them physically onto different parts of one or more display screens. (Refer section 11.3.4)