UNIT 3

Contents:

Function-Oriented Software Design: Overview of SA/SD Methodology, Structured Analysis, Structured Design, Detailed Design, Design Review.

User Interface Design: Characteristics of Good User Interface, Basic Concepts,

Introduction to Function-Oriented Software Design:

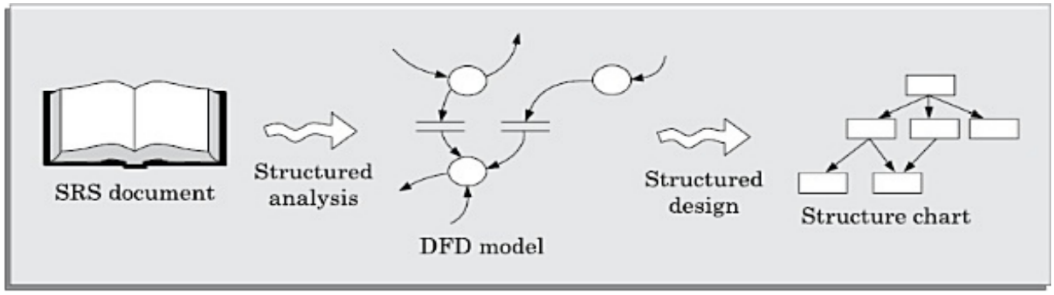
- Function-oriented design techniques were proposed nearly four decades ago.
- still very popular and are currently being used in many software development organizations. These techniques, to start with, view a system as a black-box that provides a set of services to the users of the software.
- These services are also known as the high-level functions supported by the software. During the design process, these high-level functions are successively decomposed into more detailed functions
- The term top-down decomposition i s often used to denote the successive decomposition of a set of high-level functions into more detailed functions.
- different identified functions are mapped to modules and a module structure is created.
- We shall discuss a methodology that has the essential features of several important function-oriented design methodologies.
- The design technique discussed here is called structured analysis/structured design (SA/SD) methodology.
- The SA/SD technique can b e used to perform the high-level design of a software.

Overview of SA/SD methodology.

As the name itself implies, SA/SD methodology involves carrying out two distinct activities:

- Structured analysis (SA)
- Structured design (SD).

The roles of structured analysis (SA) and structured design (SD) have been shown schematically in below figure.



- The ***structured analysis*** activity transforms the SRS document into a graphic model called the DFD model. During structured analysis, functional decomposition of the system is achieved. The purpose of structured analysis is to capture the detailed structure of the system as perceived by the user

- During ***structured design***, all functions identified during structured analysis are mapped to a module structure. This module structure is also called the high level design or the software architecture for the given problem. This is represented using a structure chart. The purpose of structured design is to define the structure of the solution that is suitable for implementation
- The high-level design stage is normally followed by a detailed design stage.

Structured Analysis

During structured analysis, the major processing tasks (high-level functions) of the system are analyzed, and t h e data flow among these processing tasks are represented graphically.
The structured analysis technique is based on the following underlying principles:

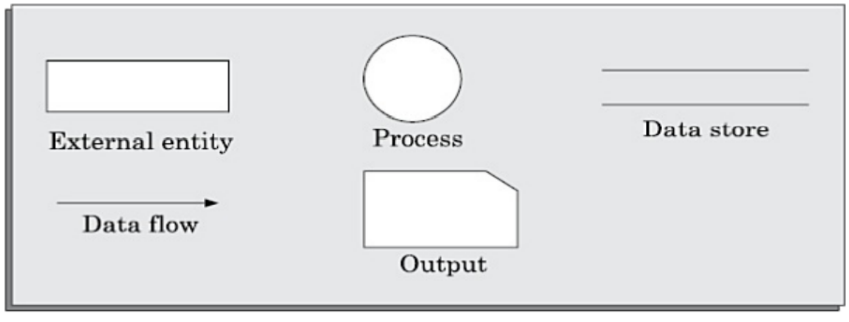
- Top-down decomposition approach.
- Application of divide and conquer principle.
- Through this each high level function is independently decomposed into detailed functions. Graphical representation of the analysis results using data flow diagrams (DFDs).

What is DFD?

- A DFD is a hierarchical graphical model of a system that shows the different processing activities or functions that the system performs and the data interchange among those functions.
- Though extremely simple, it is a very powerful tool to tackle the complexity of industry standard problems.
- A DFD model only represents the data flow aspects and does not show the sequence of execution of the different functions and the conditions based on which a function may or may not be executed.
- In the DFD terminology, each function is called a process or a bubble. each function as a processing station (or process) that consumes some input data and produces some output data.
- DFD is an elegant modeling technique not only to represent the results of structured analysis but also useful for several other applications.
- Starting with a set of high-level functions that a system performs, a DFD model represents the subfunctions performed by the functions using a hierarchy of diagrams.

Primitive symbols used for constructing DFDs

There are essentially five different types of symbols used for constructing DFDs.

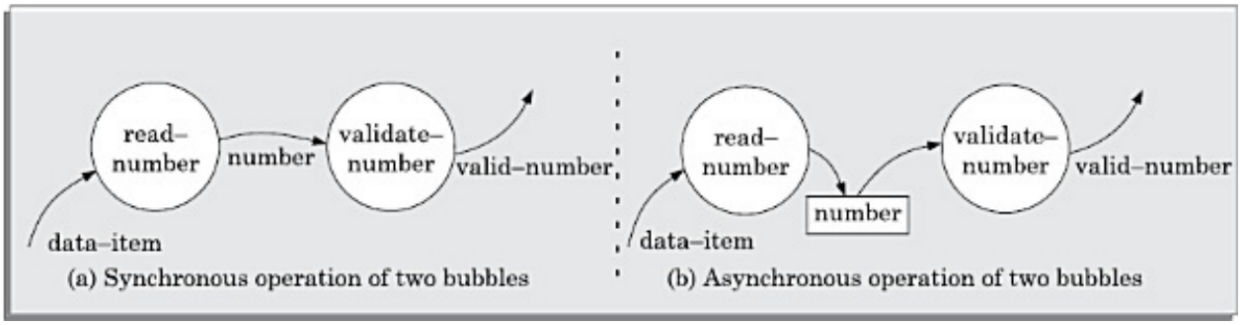


- **Function symbol:** A function is represented using a circle. This symbol is called a process or a bubble. Bubbles are annotated with the names of the corresponding functions
- **External entity symbol:** represented by a rectangle. The external entities are essentially those physical entities external to the software system which interact with the system by inputting data to the system or by consuming the data produced by the system.
- **Data flow symbol:** A directed arc (or an arrow) is used as a data flow symbol. A data flow symbol represents the data flow occurring between two processes or between an external entity and a process in the direction of the data flow arrow.
- **Data store symbol:** A data store is represented using two parallel lines. It represents a logical file. That is, a data store symbol can represent either a data structure or a physical file on disk. Connected to a process by means of a data flow symbol. The direction of the data flow arrow shows whether data is being read from or written into a data store.
- **Output symbol:** The output symbol i s as shown in Figure 6.2. The output symbol is used when a hard copy is produced.

Important concepts associated with constructing DFD models

Synchronous and asynchronous operations

- If two bubbles are directly connected by a data flow arrow, then they are synchronous. This means that they operate at the same speed.
- If two bubbles are connected through a data store, as in Figure (b) then the speed of operation of the bubbles are independent.



Data dictionary

- Every DFD model of a system must be accompanied by a data dictionary. A data dictionary lists all data items that appear in a DFD model.
- A data dictionary lists the purpose of all data items and the definition of all composite data items in terms of their component data items.
- It includes all data flows and the contents of all data stores appearing on all the DFDs in a DFD model.
- For the smallest units of data items, the data dictionary simply lists their name and their type.
- Composite data items are expressed in terms of the component data items using certain operators.

- The dictionary plays a very important role in any software development process, especially for the following reasons:
    - A data dictionary provides a standard terminology for all relevant data for use by the developers working in a project.
    - The data dictionary helps the developers to determine the definition of different data structures in terms of their component elements while implementing the design.
    - The data dictionary helps to perform impact analysis. That is, it is possible to determine the effect of some data on various processing activities and vice versa.
- For large systems, the data dictionary can become extremely complex and voluminous.
- Computer-aided software engineering (CASE) tools come handy to overcome this problem.
- Most CASE tools usually capture the data items appearing in a DFD as the DFD is drawn, and automatically generate the data dictionary.

Self Study:

- ☐ ***Data Definition***

Developing the DFD model of a system:

- The DFD model of a problem consists of many DFDs and a single data dictionary. The DFD model of a system i s constructed by using a hierarchy of DFDs.
- The top level DFD is called the level 0 DFD or the context diagram.
    - This is the most abstract (simplest) representation of the system (highest level). It is the easiest to draw and understand.
- At each successive lower level DFDs, more and more details are gradually introduced.
- To develop a higher-level DFD model, processes are decomposed into their subprocesses and the data flow among these subprocesses are identified.
- Level 0 and Level 1 consist of only one DFD each. Level 2 may contain up to 7 separate DFDs, and level 3 up to 49 DFDs, and so on.
- However, there is only a single data dictionary for the entire DFD model.

Context Diagram/Level 0 DFD:

- The context diagram is the most abstract (highest level) data flow representation of a system. It represents the entire system as a single bubble.
- The bubble in the context diagram is annotated with the name of the software system being developed.
- The context diagram establishes the context in which the system operates; that is, who are the users, what data do they input to the system, and what data they received by the system.
- The data input to the system and the data output from the system are represented as incoming and outgoing arrows.

Level 1 DFD:

- The level 1 DFD usually contains three to seven bubbles.
- The system is represented as performing three to seven important functions.

- To develop the level 1 DFD, examine the high-level functional requirements in the SRS document.
- If there are three to seven high level functional requirements, then each of these can be directly represented as a bubble in the level 1 DFD.
- Next, examine the input data to these functions and the data output by these functions as documented in the SRS document and represent them appropriately in the diagram.

Decomposition:

- Each bubble in the DFD represents a function performed by the system.
- The bubbles are decomposed into subfunctions at the successive levels of the DFD model. Decomposition of a bubble is also known as factoring o r exploding a bubble.
- Each bubble at any level of DFD is usually decomposed to anything from three to seven bubbles. Decomposition of a bubble should be carried on until a level is reached at which the function of the bubble can be described using a simple algorithm.

Self Study: Examples of DFD Models

- ☐ ***RMS Calculator***
- ☐ ***Trading House Automation System***

Structured Design

- The aim of structured design is to transform the results of the structured analysis into a structure chart.
- A structure chart represents the software architecture.
- The various modules making up the system, the module dependency (i.e. which module calls which other modules), and the parameters that are passed among the different modules.
- The structure chart representation can be easily implemented using some programming language.
- The basic building blocks using which structure charts are designed are as following:
  - ***Rectangular boxes:*** A rectangular box represents a module.
  - ***Module invocation arrows:*** An arrow connecting two modules implies that during program execution control is passed from one module to the other in the direction of the connecting arrow.
  - ***Data flow arrows:*** These are small arrows appearing alongside the module invocation arrows. represent the fact that the named data passes from one module to the other in the direction of the arrow.
  - ***Library modules:*** A library module is usually represented by a rectangle with double edges. Libraries comprise the frequently called modules.
  - ***Selection:*** The diamond symbol represents the fact that one module of several modules connected with the diamond symbol i s invoked depending on the outcome of the condition attached with the diamond symbol.

    - ***Repetition:*** A loop around the control flow arrows denotes that the respective modules are invoked repeatedly.
- In any structure chart, there should be one and only one module at the top, called the root.
- There should be at most one control relationship between any two modules in the structure chart. This means that if module A invokes module B, module B cannot invoke module A.

Flow Chart vs Structure chart:

- Flow chart is a convenient technique to represent the flow of control in a program.
- A structure chart differs from a flow chart in three principal ways:
  1. It is usually difficult to identify the different modules of a program from its flow chart representation.
  2. Data interchange among different modules is not represented in a flow chart.
  3. Sequential ordering of tasks that i s inherent to a flow chart is suppressed in a structure chart.

Transformation of a DFD Model into Structure Chart:

- Systematic techniques are available to transform the DFD representation of a problem into a module structure represented as a structure chart.
- Structured design provides two strategies to guide transformation of a DFD into a structure chart:
    - Transform analysis
    - Transaction analysis
- Normally, one would start with the level 1 DFD, transform it into module representation using either the transform or transaction analysis and then proceed toward the lower level DFDs
- At each level of transformation, it is important to first determine whether the transform or the transaction analysis is applicable to a particular DFD.

Whether to apply transform or transaction processing?

Given a specific DFD of a model, one would have to examine the data input to the diagram.
If all the data flow into the diagram are processed in similar ways (i.e. if all the input data flow arrows are incident on the same bubble in the DFD) then transform analysis is applicable. Otherwise, transaction analysis is applicable.

Transform Analysis:

- Transform analysis identifies the primary functional components (modules) and the input and output data for these components.
- The first step in transform analysis is to divide the DFD into three types of parts:
    - Input (afferent branch)
    - Processing (central transform)
    - Output (efferent branch)
- In the next step of transform analysis, the structure chart is derived by drawing one functional component each for the central transform, the afferent and efferent branches.
- These are drawn below a root module, which would invoke these modules.

- In the third step of transform analysis, the structure chart is refined by adding sub functions required by each of the high-level functional components.

Transaction Analysis:

- Transaction analysis is an alternative to transform analysis and is useful while designing transaction processing programs.
- A transaction allows the user to perform some specific type of work by using the software.
- For example, 'issue book', 'return book', 'query book', etc., are transactions.
- As in transform analysis, first all data entering into the DFD need to be identified.
- In a transaction-driven system, different data items may pass through different computation paths through the DFD.
- This is in contrast to a transform centered system where each data item entering the DFD goes through the same processing steps.
- Each different way in which input data is processed is a transaction. For each identified transaction, trace the input data to the output.
- All the traversed bubbles belong to the transaction.
- These bubbles should be mapped to the same module on the structure chart.
- In the structure chart, draw a root module and below this module draw each identified transaction as a module.

Detailed Design:

- During detailed design the pseudo code description of the processing and the different data structures are designed for the different modules of the structure chart.
- These are usually described in the form of module specifications (MSPEC). MSPEC is usually written using structured English.
- The MSPEC for the non-leaf modules describe the different conditions under which the responsibilities are delegated to the lower level modules.
- The MSPEC for the leaf-level modules should describe in algorithmic form how the primitive processing steps are carried out.
- To develop the MSPEC of a module, it is usually necessary to refer to the DFD model and the SRS document to determine the functionality of the module.

Design Review:

- After a design is complete, the design is required to be reviewed.
- The review team usually consists of members with design, implementation, testing, and maintenance perspectives.
- The review team checks the design documents especially for the following aspects:
  - ***Traceability***: Whether each bubble of the DFD can be traced to some module in the structure chart and vice versa.
  - ***Correctness***: Whether all the algorithms and data structures of the detailed design are correct.
  - ***Maintainability***: Whether the design can be easily maintained in future.
  - ***Implementation***: Whether the design can be easily and efficiently implemented.

- After the points raised by the reviewers are addressed by the designers, the design document becomes ready for implementation.

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.
- User interface part of a software product is responsible for all interactions.
- The user interface part of any software product is of direct concern to the end-users.
- No wonder then that many users often judge a software product based on its user interface
- an interface that is difficult to use leads to higher levels of user errors and ultimately leads to user dissatisfaction.
- Sufficient care and attention should be paid to the design of the user interface of any software product.
- Systematic development of the user interface is also important.
- Development of a good user interface usually takes a significant portion of the total system development effort.
- For many interactive applications, as much as 50 per cent of the total development effort is spent on developing the user interface part.
- Unless the user interface is designed and developed in a systematic manner, the total effort required to develop the interface will increase tremendously.

Characteristics of a good User Interface

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.
  - A good user interface should not require its users to memorize commands.
  - Neither should the user be asked to remember information from one screen to another
    - **Use of metaphors and intuitive command names:** Speed of learning an interface is greatly facilitated if these are based on some dayto-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.
    - **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.
    - **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.
  - The speed of learning characteristic of a user interface can be determined by measuring the training time and practice that users require before they can effectively use the software.

- ***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.
  - 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.
  - This can be achieved through careful design of the interface.
  - The most frequently used commands should have the smallest length or be available at the top of a menu.
- ***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.
  - The interface should prevent the user from entering wrong values.
- ***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.
- ***Feedback:***
  - A good user interface must provide feedback to various user actions.
  - Especially, if any user request takes more than a 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.
- ***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.
- The skill level of users improves as they keep using a software product and they look for commands to suit their skill levels.

- ***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:

User Guidance and Online 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.

1. ***Online 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 online 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.
2. ***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.
3. ***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.

Mode-based and modeless interfaces:

- 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.
- A mode-based interface can be represented using a state transition diagram.

Graphical User Interface versus Text-based User Interface:

- In a GUI multiple windows with different information can simultaneously be displayed on the user screen. user has the flexibility to simultaneously interact with several related items at any time
- Iconic information representation and symbolic information manipulation is possible in a GUI.
- 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.
- A GUI requires special terminals with graphics capabilities for running and also requires special input devices such a mouse.
- 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, They have become affordable.

Types of User Interfaces:

- Broadly speaking, user interfaces can be classified into the following three categories:
  - Command language-based interfaces.
  - Menu-based interfaces.
  - Direct manipulation interfaces.
- Each of these categories of interfaces has its own characteristic advantages and disadvantages.

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.

- The command language interface allows for the most efficient command issue procedure requiring minimal typing.
- a command language-based interface can be implemented even on cheap alphanumeric terminals.
- 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
- command language-based interfaces suffer from several drawbacks.
- command language-based interfaces are difficult to learn and require the user to memorize the set of primitive commands.
- Most users make errors while formulating commands.
- All interactions with the system are through a key-board and cannot take advantage of effective interaction devices such as a mouse.

Issues in designing a command language based interface:

- The designer has to decide what mnemonics (command names) to use for the different commands.
- The designer has to decide whether the users will be allowed to redefine the command names to suit their own preferences.
- The designer has to decide whether it should be possible to compose primitive commands to form more complex commands.

Menu-Based Interfaces:

- 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.
- In a menu-based interface the typing effort is minimal.
- A major challenge in the design of a menu-based interface is to structure a large number of menu choices into manageable forms.
- 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.
        - 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.
    - ***Walking menu:***
        - Walking menu is very commonly used to structure a large collection of menu items.
        - 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.

- ***Hierarchical menu:***
  - This type of menu is suitable for small screens with limited display area such as that in mobile phones.
  - 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.

Direct Manipulation Interfaces:

- Direct manipulation interfaces present the interface to the user in the form of visual models.
- Direct manipulation interfaces are sometimes called 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 recognised by the users very easily, and that icons are language independent.
- However, experienced users find direct manipulation interfaces very useful too.
- Also, it is difficult to give complex commands using a direct manipulation interface.

User Interface Design Methodology

- At present, no step-by-step methodology is available which can be followed by rote to come up with a good user interface.
- What we present in this section is a set of recommendations which you can use to complement your ingenuity.

A GUI Design Methodology:

- The GUI design methodology we present here is based on the seminal work of Frank Ludolph.
- Our user interface 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.

Examining the use case model

- The starting point for GUI design is the use case model.
- This captures the important tasks the users need to perform using the software.
- Metaphors help in interface development at lower effort and reduced costs for training the users.

- 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.

Task and object Modeling:

- A task is a human activity intended to achieve some goals.
- Examples of task goals can be as follows:
    - 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.
- A user object model is a model of business objects which the end-users believe that they are interacting with.
- The objects in a library software may be books, journals, members, etc.

Metaphor selection:

- 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.

Interaction design and rough layout

- The interaction design involves mapping the subtasks into appropriate controls, and other widgets such as forms, text box, etc.
- This involves making a choice from a set of available components that would best suit the subtask.
- Rough layout concerns how the controls, and other widgets to be organized in windows.

Detailed presentation and graphics design

- Each window should represent either an object or many objects that have a clear relationship to each other.
- At one extreme, each object view could be in its own window. But, this is likely to lead to too much window opening, closing, moving, and resizing.
- At the other extreme, all the views could be placed in one window side-by-side, resulting in a very large window.
- This would force the user to move the cursor around the window to look for different objects.

GUI construction

- Some of the windows have to be defined as modal dialogs.
- When a window is a modal dialog, no other windows in the application are accessible until the current window is closed.
- When a modal dialog is closed, the user is returned to the window from which the modal dialog was invoked.
- Modal dialogs are commonly used when an explicit confirmation or authorisation step is required for an action.

User interface inspection

- Nielson studied common usability problems and built a check list of points which can be easily checked for an interface. The following checklist is based on the work of Nielson:
    - Visibility of the system status
    - Match between the system and the real world
    - Undoing mistakes
    - Consistency
    - Recognition rather than recall
    - Support for multiple skill levels
    - Aesthetic and minimalist design
    - Help and error messages
    - Error prevention