SOFTWARE ENGINEERING
UNIT-3

1. **Function-Oriented Software Design:** Overview of SA/SD Methodology
2. Structured Analysis,
3. Developing the DFD Model of aSystem,
4. Structured Design,
5. Detailed Design,
6. DesignReview,
7. **User Interface Design:** Characteristics of Good UserInterface,
8. BasicConcepts,
9. Types of UserInterfaces,
10. Fundamentals of Component-based GUIDevelopment,
11. A User Interface Design Methodology

FUNCTION-ORIENTED SOFTWARE DESIGN

- Function-oriented design techniques were proposed nearly four decadesago.
- These techniques are at the present time still very popular and are currently being used in many software developmentorganisations.
- These techniques, to start with, view a system as a black-box that provides a set of services to the users of thesoftware.
- These services provided by a software (e.g., issue book, search book, etc.,) for a Library Automation Software to its users 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 detailedfunctions.
- The term top-down decomposition is often used to denote the successive decomposition of a set of high-level functions into more detailedfunctions.
- After top-down decomposition has been carried out, the different identified functions are mapped to modules and a module structure iscreated.
- This module structure would possess all the characteristics of a good design ident ified in the lastchapter.
- The SA/SD technique can b e used to perform the high-level design of a software. The details of SA/SD technique are discussedfurther.

1. 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 Figure6.1.

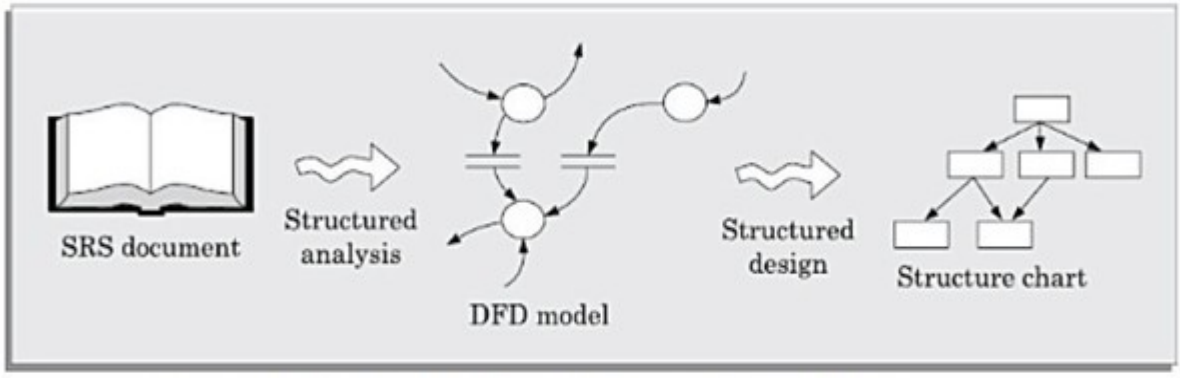


Figure 6.1: Structured analysis and structured design methodology.

- During structured analysis, the SRS document is transformed into a data flow diagram (DFD)model.
- During structured design, the DFD model is transformed into a structurechart.

- As shown in Figure 6.1, the structured analysis activity transforms the SRS document into a graphic model called the DFDmodel.
- **During structured analysis**, functional decomposition of the system is achieved. That is, each function that the system needs to perform is analysed and hierarchically decomposed into more detailedfunctions.
- **During structured design**, all functions identified during structured analysis are mapped to a modulestructure.
- This **module structure** is also called the hig-hlevel design or the software architecture for the given problem.
- This is represented using a **structurechart**.
- The high-level design stage is normally followed by a detailed designstage.
- During the **detailed design** stage, the algorithms and data structures for the individual modules are designed.
- The detailed design can directly be implemented as a working system using a conventional programminglanguage.
- It is important to understand that the purpose of structured analysis is to capture the detailed structure of the system as perceived by the user, whereas the purpose of structured design is to define the structure of the solution that is suitable for implementation in some programminglanguage.
- The results of structured analysis can therefore, be easily understood by the user. In fact, the different functions and data in structured analysis are named using the user's terminology. The user can therefore even review the results of the structured analysis to ensure that it captures all hisrequirements.
- In the following section, w e first discuss how to carry out structured analysis to construct the DFD model. Subsequently, we discuss how the DFD model can be transformed into structureddesign.

2 STRUCTURED ANALYSIS

- We have already mentioned that during structured analysis, the major processing tasks (high-level functions) of the system are analysed, and the data flow among these processing tasks are representedgraphically.
- The structured analysis technique is based on the following underlyingprinciples:
- Top-down decompositionapproach.
- Application of divide and conquer principle. Through this each highlevel function is independently decomposed into detailed functions. Graphical representation of the analysis results using data flow diagrams(DFDs).
- DFD representation of a problem, as we shall see shortly, is very easy to construct. Though extremely simple, it is a very powerful tool to tackle the complexity of industry standardproblems.
- 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.
- Please note that 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 beexecuted.

3. Data Flow Diagrams(DFDs)

- The DFD (also known as the bubble chart) is a simple graphical formalism that can be used to represent a system in terms of the input data to the system, various processing carried out on those data, and the output data generated by thesystem.
- The main reason why the DFD technique is so popular is probably because of the fact that DFD is a very simple formalism— **it is simple to understand anduse**.
- A DFD model uses a very limited number of primitive symbols (shown in Figure 6.2) to represent the functions performed by a system and the **data flow among thesefunctions**.

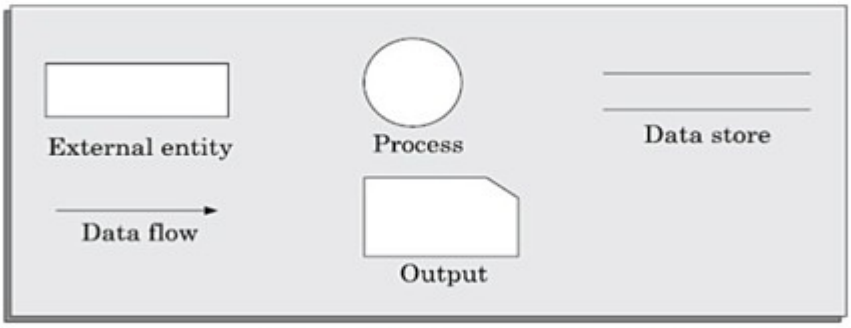


Figure 6.2: Symbols used for designing DFDs.

- Starting with a set of high-level functions that a system performs, a DFD model represents the sub-functions performed by the functions using a hierarchy ofdiagrams.
- Human mind is such that it can **easily understand any hierarchical model of a system** —because in a hierarchical model, starting with a very abstract model of a system, various details of the system are slowly introduced through different levels of the hierarchy.
- The DFD technique is also based on a very simple set of intuitive concepts and rules.

Primitive symbols used for constructing DFDs: There are essentially five different types of symbols used for constructing DFDs. These primitive symbols are depicted in Figure6.2.

- The meaning of these symbols are explained as follows: **Figure 6.2:** Symbols used for designingDFDs.
- **1. Function symbol:** A function is represented using acircle.

This symbol is called a process or a bubble. Bubbles are annotated with the names of the corresponding functions (see Figure 6.3).

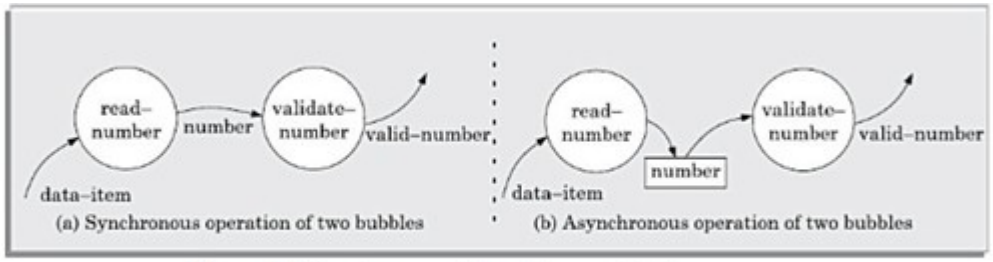
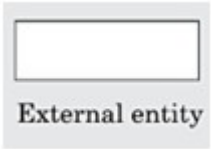


Figure 6.3: Synchronous and asynchronous data flow.

- **2. External entity symbol:** An external entity such as a librarian, a library member, etc. is represented by arectangle.



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.

In addition to the human users, the external entity symbols can be used to **represent external hardware and software** such as another application software that would interact with the software beingmodeled.

- **3. Data flow symbol:** A directed arc (or an arrow) is used as a data flowsymbol.

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.

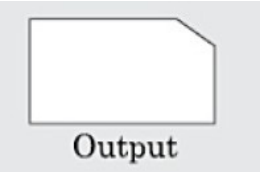
Data flow symbols are usually annotated with the corresponding data names. For example the DFD in Figure 6.3(a) shows three data flows—the data item number flowing from the process read-number to validate-number, data item flowing into read-number, and valid-number flowing out ofvalidate-number.

- **4. Data store symbol:** A data store is represented using two parallellines.

It represents a logical file. That is, a data store symbol can represent either a data structure or a physical file on disk. Each data store is 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. An arrow flowing in or out of a data store implicitly represents the entire data of the data store and hence arrows connecting t o a data store need not be annotated with the name of the corresponding data items. As an example of a data store, number is a data store in Figure6.3(b).

- **5. Output symbol:** The output symbol is as shown in Figure 6.2. The output symbol is used when a hard copy isproduced.



- **Important concepts associated with constructing DFD models** Before we discuss how to construct the DFD model of a system, let us discuss some important concepts associated withDFDs:

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. An example of such an arrangement is shown in Figure6.3(a).

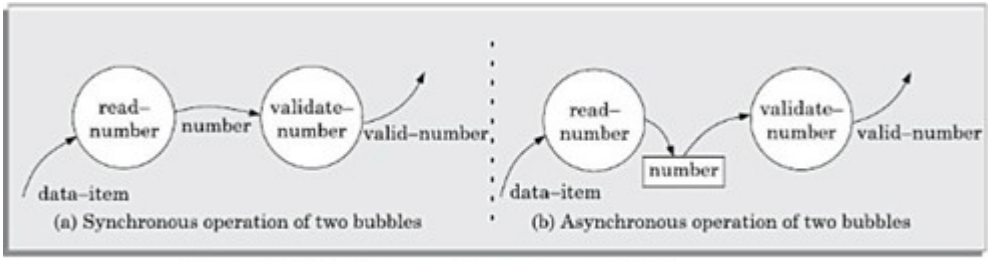


Figure 6.3: Synchronous and asynchronous data flow.

- Here, the validate-number bubble can start processing only after the read number bubble has supplied data to it; and the read-number bubble has to wait until the validate-number bubble has consumed itsdata.
- However, if two bubbles are connected through a data store, as in Figure 6.3(b) then the speed of operation of the bubbles are independent. This statement can be explained using the following reasoning. The data produced by a producer bubble gets stored in the data store. It is therefore possible that the producer bubble stores several pieces of data items, even before the consumer bubble consumes any ofthem.

Data dictionary:

- Every DFD model of a system must be accompanied by a datadictionary.
- A data dictionary lists all data items that appear in a DFD model. The data items listed include all data flows and the contents of all data stores appearing on all the DFDs in a DFDmodel.
- The DFD model of a system typically consists of several DFDs, viz., level 0 DFD, level1 DFD, level 2 DFDs, etc., as shown in Figure6.
- However, a single data dictionary should capture all the data appearing in all the DFDs constituting the DFD model of a system. 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.

- For example, a data dictionary entry may represent that the data grossPay consists of the components regularPay andovertimePay.
  grossP ay = regularP ay + overtimePay
- 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 operators using which a composite data item can be expressed in terms of its component dataitems.
- The dictionary plays a very important role in any software development process, especially for the followingreasons:
- A data dictionary provides a standard terminology for all relevant data for use by the developers working in aproject.
- The data dictionary helps the developers to determine the definition of different data structures in terms of their component elements while implementing thedesign.
- The data dictionary helps to perform impact analysis. That is, it is possible to determine the effect of some data on various processingactivities.
- For large systems, the data dictionary can become extremely complex andvoluminous.
- Moderate-sized projects can have thousands of entries in the data dictionary. It becomes extremely difficult to maintain a voluminous dictionary manually. Computer-aided software engineering (CASE) tools come handy to overcome thisproblem.
- Most CASE tools usually capture the data items appearing in a DFD as the DFD is drawn, and automatically generate the data dictionary. As a result, the designers do not have to spend almost any effort in creating the datadictionary.
- These CASE tools also support some query language facility to query about the definition and usage of dataitems.
- For example, queries may be formulated to determine which data item affects which processes, or a process affects which data items, or the definition and usage of specific data items, etc. Query handling is facilitated by storing the data dictionary in a relational database management system(RDBMS).

Data definition :

- Composite data items can be defined in terms of primitive data items using the following data definitionoperators.

+ : denotes composition of two data items, e.g. a+b represents data a andb.

[,,] : represents selection, i.e. any one of the data items listed inside the square bracket can occur For example, [a,b] represents either a occurs or b occurs.

() : the contents inside the bracket represent optional data which may or may not appear. a+(b) represents either a or a+boccurs.

{} : represents iterative data definition, e.g. *{*name*}*5 represents five namedata. *{*name*}** represents zero or more instances of name data.

= : represents equivalence, e.g. a=b+c means that a is a composite data item comprising of both b andc.

/**/ : Anything appearing within /* and */ is considered ascomment.

DEVELOPING THE DFD MODEL OF ASYSTEM

- A DFD model of a system graphically represents how each input data is transformed to its corresponding output data through a hierarchy ofDFDs.
- The DFD model of a problem consists of many of DFDs and a single datadictionary.
- TheDFDmodelofa systemisconstructedbyusinga hierarchyofDFDs (see Figure6.4).

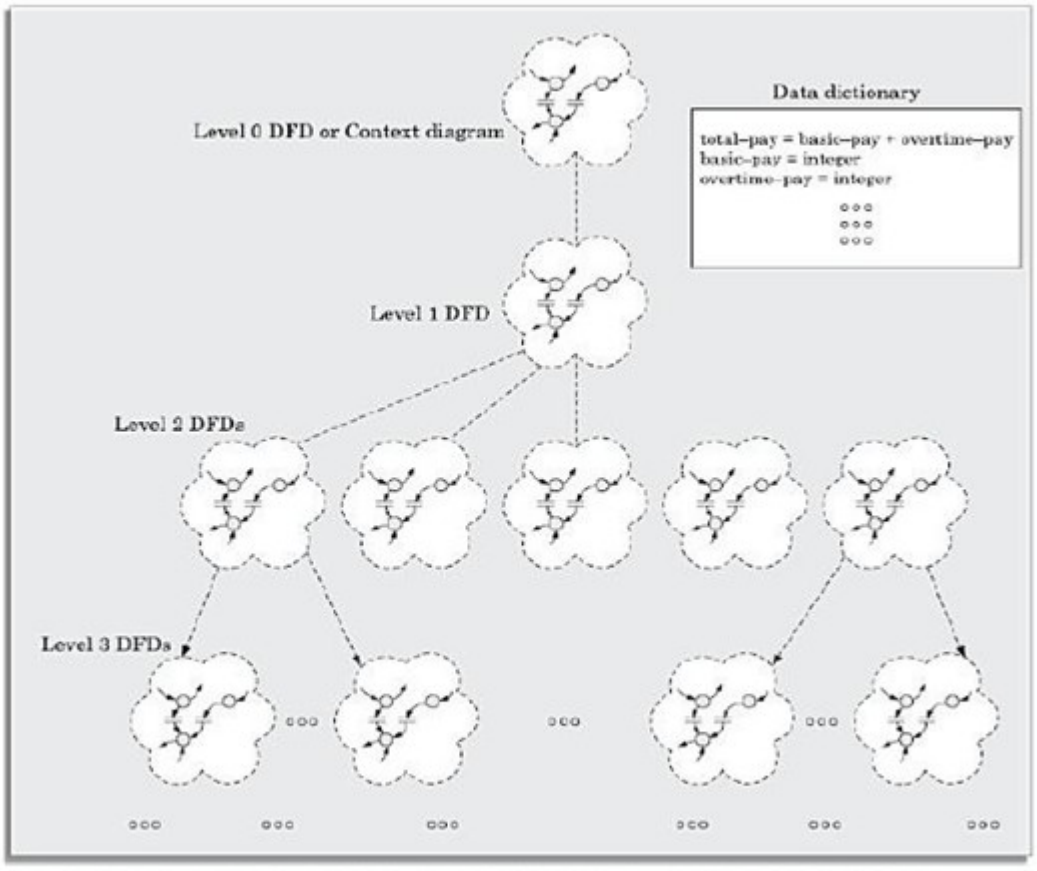


Figure 6.4: DFD model of a system consists of a hierarchy of DFDs and a single data dictionary.

- 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 DFD s, more and more details are graduallyintroduced.
- To develop a higher-level DFD model, processes are decomposed into their sub-processes and the data flow among these sub-processes areidentified.
- To develop the data flow model of a system, first the most abstract representation (highest level) of the problem is to be worked out. Subsequently, the lower level DFDs are developed. Level 0 and Level 1 consist of only one DFDeach.
- Level 2 may contain up to 7 separate DFDs, and level 3 up to 49 DFDs, and soon.
- However, there is only a single data dictionary for the entire DFD model. All the data names appearing in all DFDs are populated in the data dictionary and the data dictionary contains the definitions of all the dataitems.

Context Diagram :

- The context diagram is the most abstract (highest level) data flow representation of a system.
- It represents the entire system as a singlebubble.
- The bubble in the context diagram is annotated with the name of the software system being developed. (usually anoun).
- This is the only bubble in a DFD model, where a noun is used for naming thebubble.
- The bubbles at all other levels are annotated with verbs according to the main function performed by thebubble.
- This is expected since the purpose of the context diagram is to capture the context of the system rather than its functionality. As an example of a context diagram, consider the context diagram a software developed to automate the book keeping activities of a supermarket (see Figure 6.10). The context diagram has been labelled as 'Supermarketsoftware'.

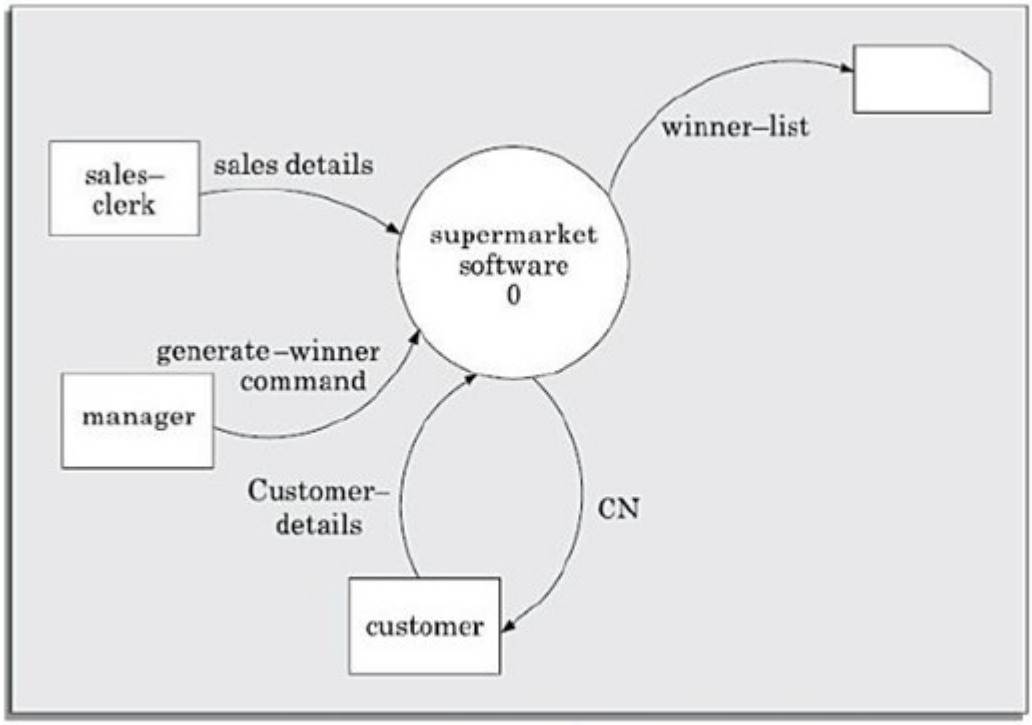


Figure 6.10: Context diagram for Example 6.3.

- **Figure 6.4:** DFD model of a system consists of a hierarchy of DFDs and a single data dictionary.

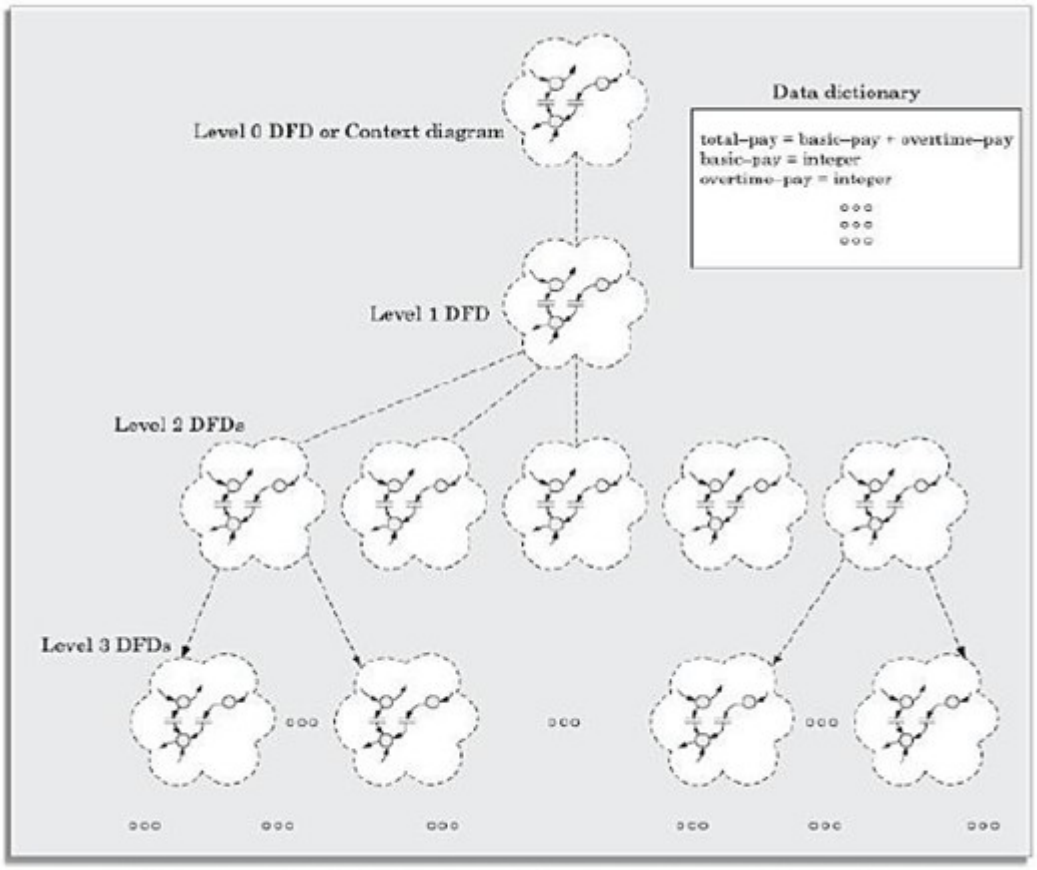


Figure 6.4: DFD model of a system consists of a hierarchy of DFDs and a single data dictionary.

- The context diagram establishes the context in which the system operates; thatis,
  - who are theusers,
  - what data do they input to the system,and
  - what data they received by thesystem.
- The name context diagram of the level 0 DFD is justified because it represents the context in which the system would exist; that is, the external entities who would interact with thesystem
- The various external entities with which the system interacts and the data flow occurring between the system and the external entities arerepresented.
- The data input to the system and the data output from the system are represented as incoming and outgoingarrows.
- These data flow arrows should be annotated with the corresponding datanames.
- **To develop the context diagram of the system, we have to analyse the SRS document to identify the different types of users**
- Here, the term users of the system also includes any external systems which supply data to or receive data from thesystem.

Level 1 DFD :

- The level 1 DFD usually contains three to sevenbubbles.
- That is, the system is represented as performing three to seven importantfunctions.
- To develop the level 1 DFD, examine the high-level functional requirements in the SRSdocument.
- If there are three to seven high level functional requirements, then each of these can be directly represented as a bubble in the level 1DFD.
- 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 thediagram.
- What if a system has more than seven high-level requirements identified in the SRS document? In this case, some of the related requirements have to be combined and represented as a single bubble i n the level 1DFD.
- These can be split appropriately in the lower DFD levels. If a system has less than three high-level functional requirements, then some of the high-level requirements need to be split into their sub-functions so that we have roughly about five to seven bubbles represented on the diagram. Level 1 DFDs Examples are shown in 6.1 to6.4.

Decomposition :

- Each bubble in the DFD represents a function performed by the system.
- The bubbles are decomposed intosub-functions.
- Decomposition of a bubble is also known as factoring or exploding abubble.
- Each bubble at any level of DFD is usually decomposed to anything three to seven bubbles.
- For example, if a bubble is decomposed to just one bubble or two bubbles, then this decomposition becomes repetitive. On the other hand, too many bubbles (i.e. more than seven bubbles) at any level of a DFD makes the DFD model hard tounderstand.
- 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 simplealgorithm.

Developing the DFD model of a system more systematically.

- **1. Construction of context diagram:** Examine the SRS document todetermine:
  - Different high-level functions that the system needs toperform.
  - Data input to every high-levelfunction.
  - Data output from every high-levelfunction.
  - Interactions (data flow) among the identified high-levelfunctions.
  - Represent these aspects of the high-level functions in a diagrammatic form. This would form the top-level data flow diagram (DFD), usually called the DFD0.
- **2. Construction of level 1 diagram:** Examine the high-level functions described in the SRSdocument.
  - If there are three to seven high-level requirements in the SRS document, then represent each of the high-level function in the form of abubble.
  - If there are more than seven bubbles, then some of them have to becombined.
  - If there are less than three bubbles, then some of these have to besplit.
- **3. Construction of lower-level diagrams:** Decompose each high-level function into its constituent sub-functions through the following set ofactivities:
  - Identify the different sub-functions of the high-levelfunction.

- Identify the data input to each of thesesub-functions.
- Identify the data output from each of thesesub-functions.
- Identify the interactions (data flow) among thesesub-functions.
- Represent these aspects in a diagrammatic form using aDFD.
- Recursively repeat Step 3 for each sub-function until a sub-function can be represented by using a simplealgorithm.

Numbering of bubbles

- It is necessary to number the different bubbles occurring in theDFD.
- These numbers help in uniquely identifying any bubble in the DFD from its bubble number.
- The bubble at the context level is usually assigned the number 0 to indicate that it is the 0 levelDFD.
- Bubbles at level 1 are numbered, 0.1, 0.2, 0.3,etc.
- When a bubble numbered x is decomposed, its children bubble are numbered x.1, x.2, x.3,etc.
- In this numbering scheme, by looking at the number of a bubble we can unambiguously determine its level, its ancestors, and itssuccessors.

Balancing DFDs :

- The DFD model of a system usually consists of many DFDs that are organized in a hierarchy.
- In this context, a DFD is required to be balanced with respect to the corresponding bubble of the parentDFD.
- The data that flow into or out of a bubble must match the data flow at the next level of DFD. This is known as balancing aDFD.
- Balancing a DFD shown in Figure6.5.

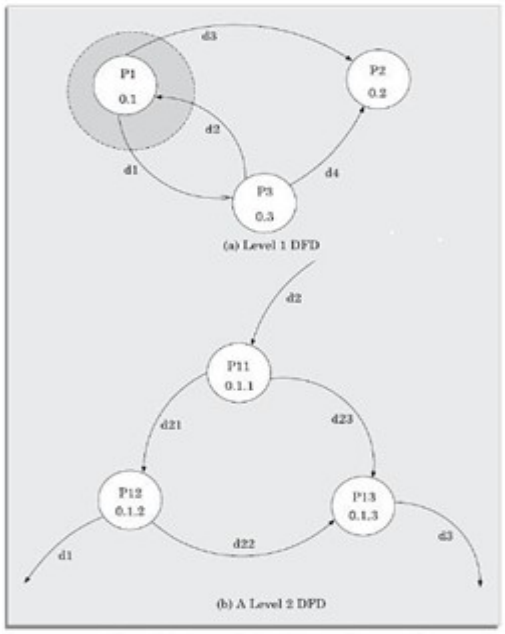


Figure 6.5: An example showing balanced decomposition.

- In the level 1 DFD, data items d1 and d3 flow out of the bubble 0.1 and the data item d2 flows into the bubble 0.1 (shown by the dottedcircle).
- In the next level, bubble 0.1 is decomposed into three DFDs(0.1.1,0.1.2,0.1.3).
- The decomposition is balanced, as d1 and d3 flow out of the level 2 diagram and d2 flows in.
- Dangling arrows (d1,d2,d3) represent the data flows into or out of adiagram.

Illustration 1. A book can be searched in the library catalog by inputting its name. If the book is available in the library, then the details of the book are displayed. If the book is not listed in the catalog, then an error message is generated. While developing the DFD model for this simple problem, many beginners commit the mistake of drawing an arrow (as shown in Figure 6.6) to indicate that the error function is invoked after the search book. But, this is a control information and should not be shown on the DFD.

Figure 6.6: It is incorrect to show control information on a DFD.

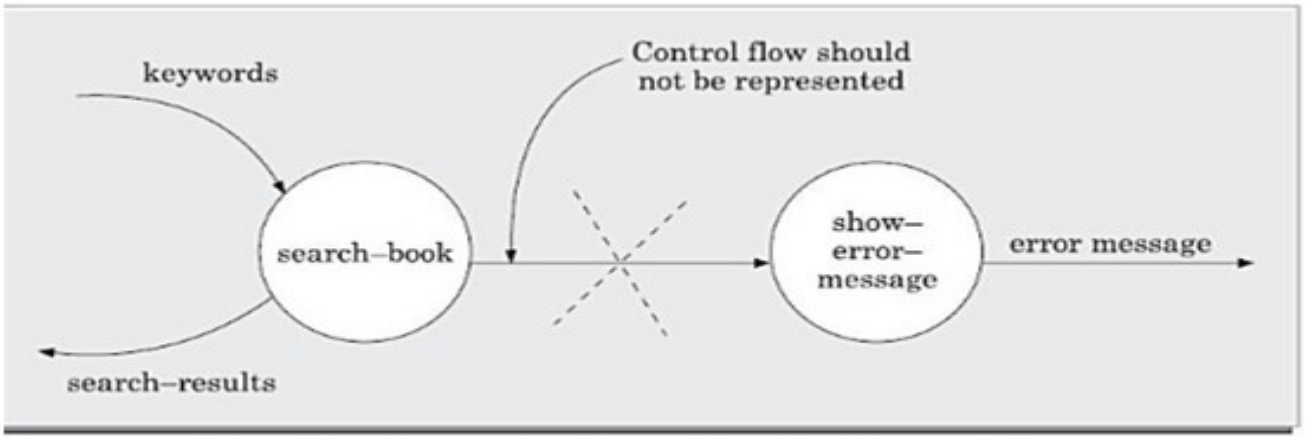


Figure 6.6: It is incorrect to show control information on a DFD.

Example-1 (RMS Calculating Software)

- A software system called RMS calculating software would read three integral numbers from the user in the range of –1000 and +1000 and would determine the root mean square (RMS) of the three input numbers and displayit.
- In this example, the context diagram is simple todraw.
- The system accepts three integers from the user and returns the result to him.
- This has been shown in Figure6.8(a).

Figure 6.8: Context diagram, level 1, and level 2 DFDs for Example 6.1.

/* ADDITIONAL INFORMATION- START*/

Example-2 (Tic-Tac-Toe Computer Game)

- Tic-tac-toe is a computer game in which a human player and the computer make alternate moves on a 3 × 3square.
- A move consists of marking a previously unmarkedsquare.
- The player who is first to place three consecutive marks along a straight line (i.e., along a row, column, or diagonal) on the squarewins.
- As soon as either of the human player or the computer wins, a message congratulating the winner should bedisplayed.
- If neither player manages to get three consecutive marks along a straight line, and all the squares on the board are filled up, then the game isdrawn.
- The computer always tries to win agame.
- The context diagram and the level 1 DFD are shown in Figure6.9.

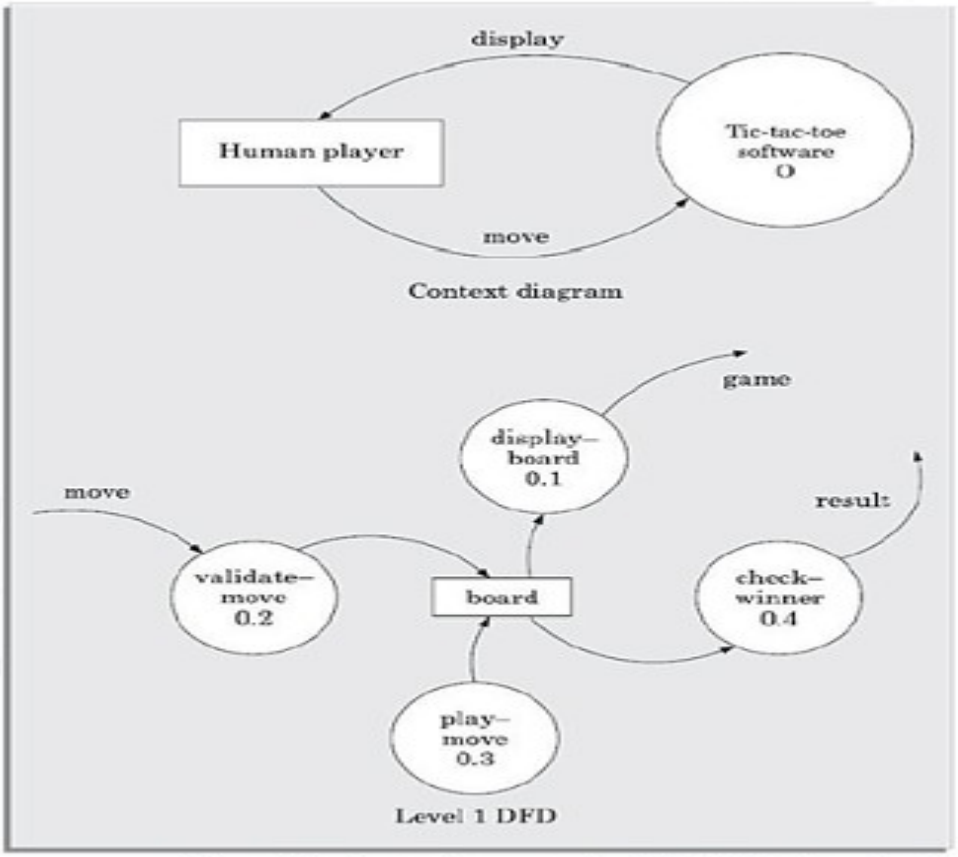


Figure 6.9: Context diagram and level 1 DFDs for Example 6.2.

Example-3 (Supermarket Prize Scheme)

- A super market needs to develop a software that would help it to automate a scheme that it plans to introduce to encourage regularcustomers.
- In this scheme, a customer would have first register by supplying his/her residence address, telephone number, and the driving licensenumber.
- Each customer who registers for this scheme is assigned a unique customer number (CN) by the computer. A customer can present his CN to the check out staff when he makes anypurchase.
- In this case, the value of his purchase is credited against hisCN.
- At the end of each year, the supermarket intends to award surprise gifts to 10 customers who make the highest total purchase over theyear.
- Also, it intends to award a 22 caret gold coin to every customer whose purchase exceeded Rs.10,000.
- The entries against the CN are reset on the last day of every year after the prize winners' lists are generated.

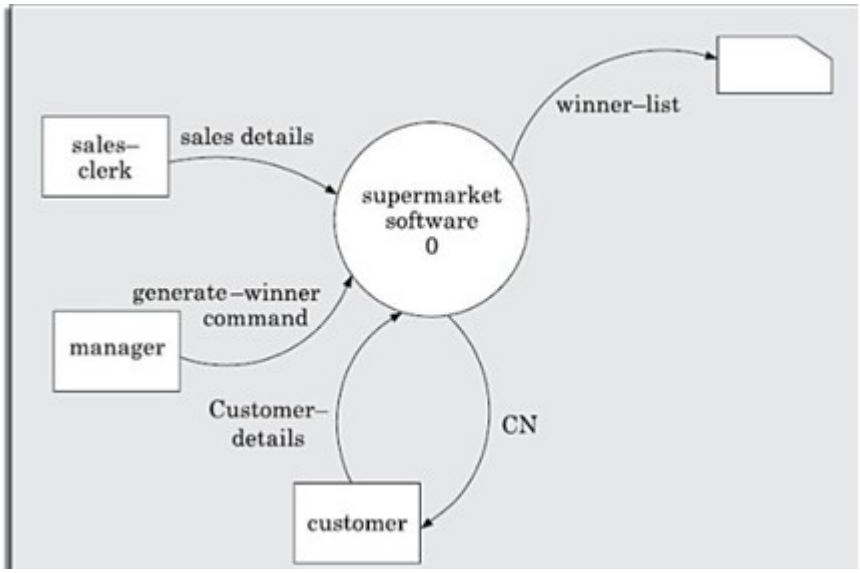


Figure 6.10: Context diagram for Example-3.

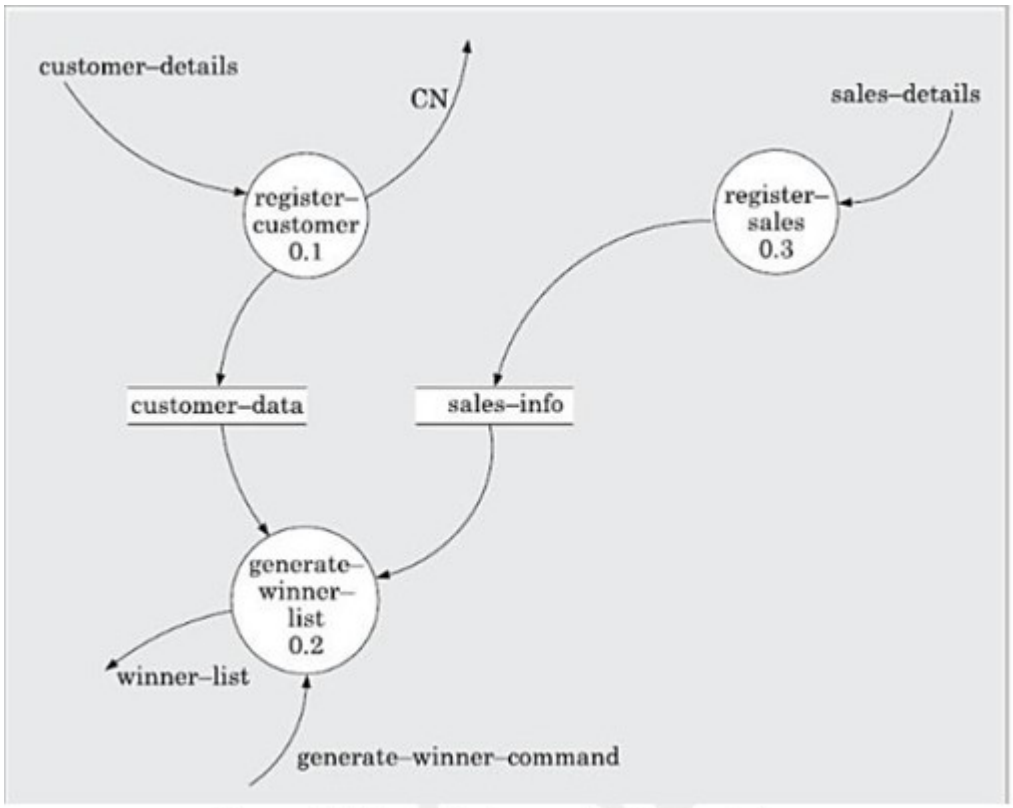


Figure 6.11: Level 1 diagram for Example-3.

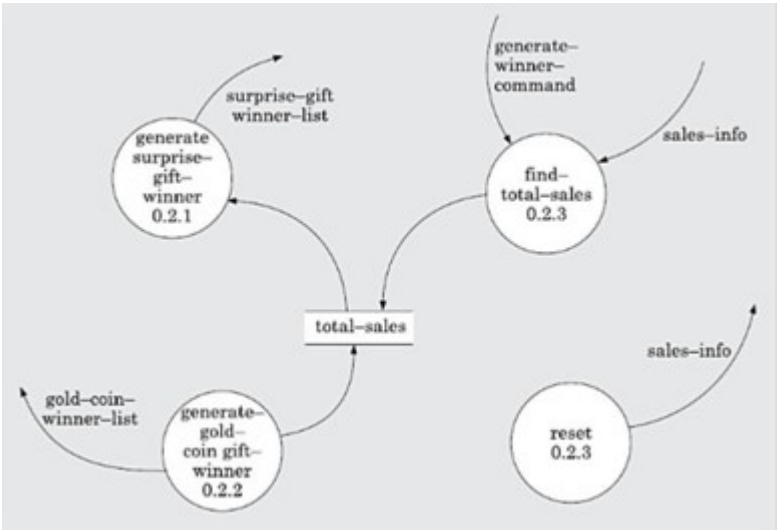


Figure 6.12: Level 2 diagram for Example-3.

Example 4 (Trading-house Automation System (TAS))

- A trading house wants us to develop a computerized system that would automate various bookkeeping activities associated with itsbusiness.
- The following are the important features of the system to bedeveloped:
- The trading house has a set of regular customers.
- The customers place orders with it for various kinds ofcommodities.

- The trading house maintains the names and addresses of its regularcustomers.
- Each of these regular customers should be assigned a unique customer identification number (CIN) by thecomputer.
- The customers quote their CIN on every order theyplace.
- Once order is placed, as per current practice, the accounts department of the trading house first checks the credit-worthiness of thecustomer.
- The credit-worthiness of the customer is determined by analysing the history of his payments to different bills sent to him in thepast.
- After automation, this task has be done by thecomputer.
- If a customer is not credit-worthy, his orders are not processed any further and an appropriate order rejection message is generated for thecustomer.
- If a customer is credit-worthy, the items that he has ordered are checked against the list of items that the trading house dealswith.
- The items in the order which the trading house does not deal with, are not processed any further and an appropriate apology message for the customer for these items isgenerated.
- The items in the customer's order that the trading house deals with are checked for availability in theinventory.
- If the items are available in the inventory in desired quantity,then:
  – A bill is with the forwarding address of the customer isprinted.
  – A material issue slip is printed. The customer can produce this material issue slip at the store house and take delivery of theitems.
  – Inventory data is adjusted to reflect the sale to thecustomer.
- If any of the ordered items are not available in the inventory in sufficient quantity to satisfy the order, then these out-of-stock items along with the quantity ordered by the customer and the CIN are stored in a "pending-order" file for further processing to be carried out when the purchase department issues the "generate indent"command.
- The purchase department should be allowed to periodically issue commands to generate indents.
- When a command to generate indents is issued, the system should examine the "pending-order" file to determine the orders that are pending and determine the total quantity required for each of the items. It should find out the addresses of the vendors who supply these items by examining a file containing vendor details and then should print out indents to thesevendors.
- The system should also answer managerial queries regarding the statistics of different items sold over any given period of time and the corresponding quantity sold and the price realised.
- The context diagram for the trading house automation problem is shown in Figure 6.13. The level 1 DFD in Figure6.14.

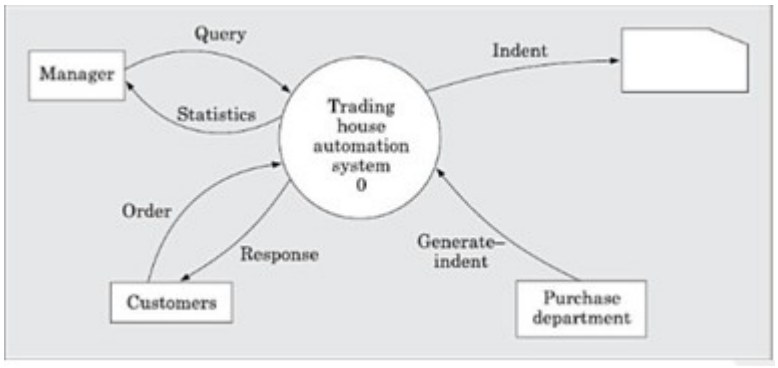


Figure 6.13: Context diagram for Example- 4.

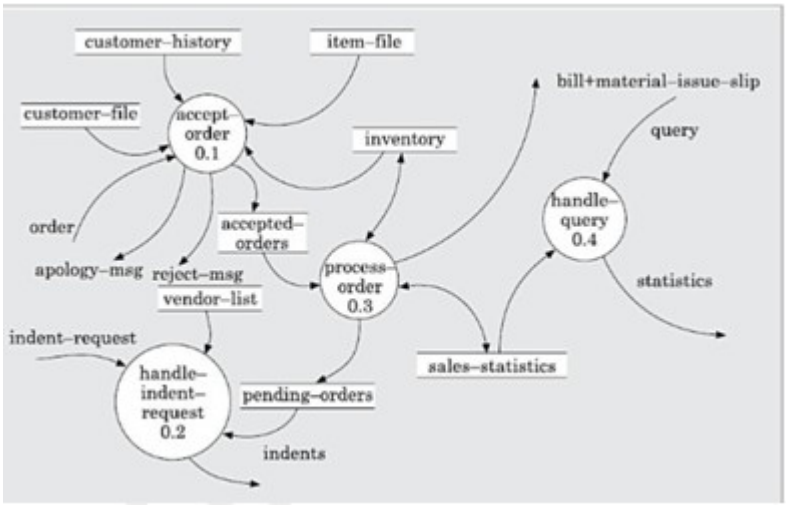


Figure 6.14: Level 1 DFD for Example-4.

Example-5 (Personal Library Software) Perform structured analysis for the personal library software of Example-5.
The context diagram is shown in Figure 6.15.

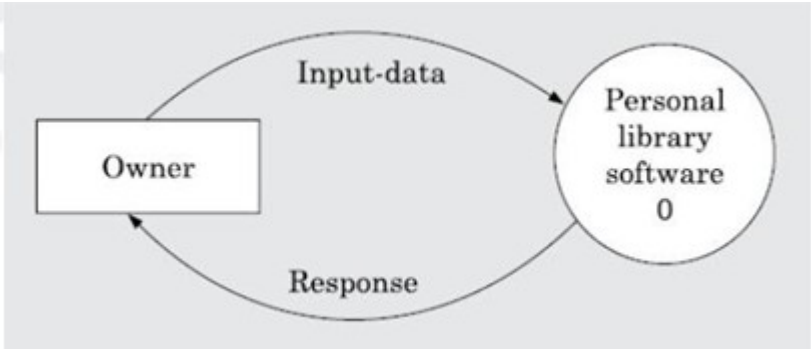


Figure 6.15: Context diagram for Example-5.

The level 1 DFD is shown in Figure 6.16.

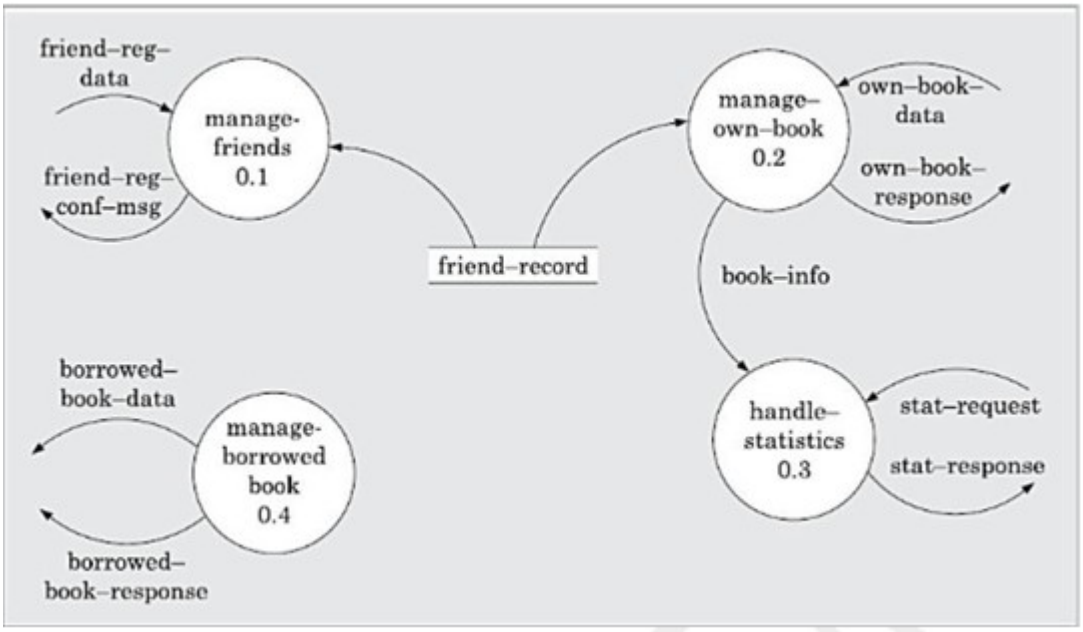


Figure 6.16: Level 1 DFD for Example-5.

The level 2 DFD for the manageOwnBook bubble is shown in Figure 6.17.

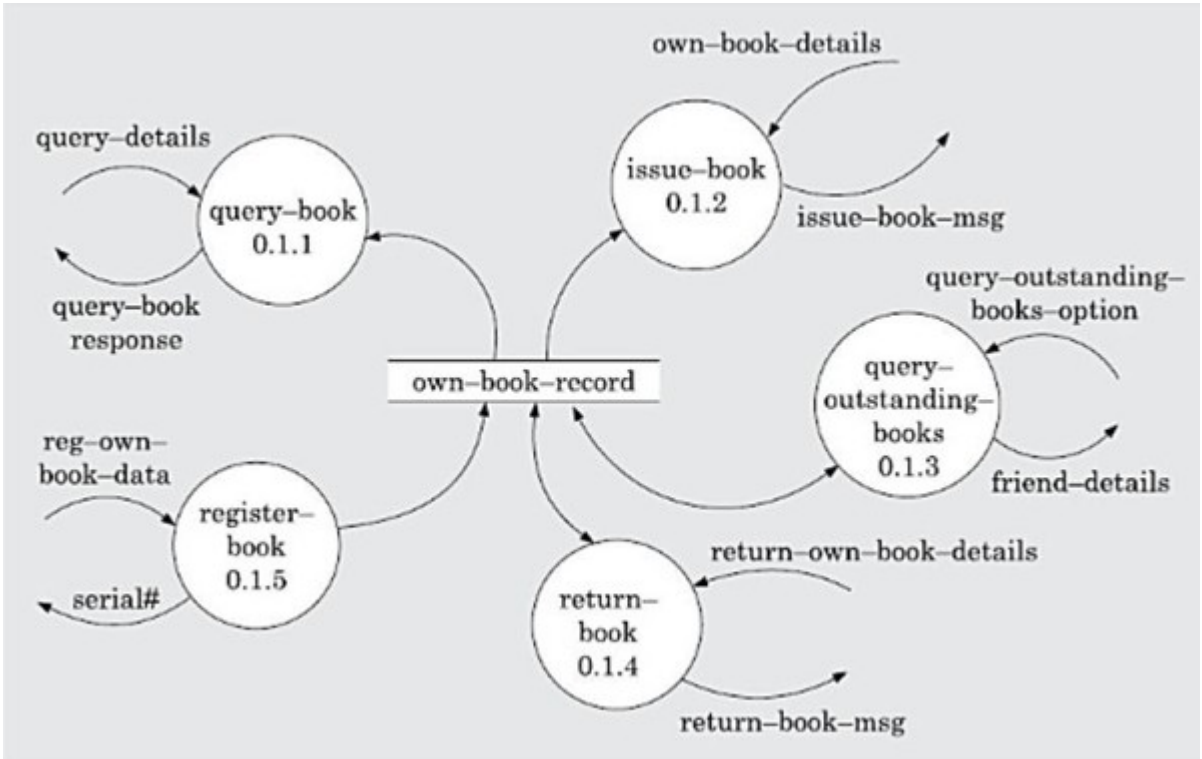


Figure 6.17: Level 2 DFD for Example-5.

4. STRUCTUREDDESIGN

- The aim of structured design is to transform the results of the structured analysis (that is, the DFD model) into a **structurechart**.
- **A structure chart represents the softwarearchitecture.**
- 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 main focus in a structure chart representation is on module structure of a software and the interaction among the different modules, the procedural aspects (e.g. how a particular functionality is achieved) are notrepresented.
- The basic building blocks using which structure charts are designed are asfollowing:
- **Rectangular boxes:** A rectangular box represents a module. Usually, every rectangular box is annotated with the name of the module itrepresents.
- **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 connectingarrow.

    - However, just by looking at the structure chart, we cannot say whether a modules calls another module just once or manytimes.
    - Also, just by looking at the structure chart, we cannot tell the order in which the different modules areinvoked.
- **Data flow arrows:** These are small arrows appearing alongside the module invocation arrows. The data flow arrows are annotated with the corresponding data name. Data flow arrows represent the fact that the named data passes from one module to the other in the direction of thearrow.
- **Library modules:** A library module is usually represented by a rectangle with double edges. Libraries comprise the frequently called modules. Usually, when a module is invoked by many other modules, it is made into a librarymodule.
- **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 diamondsymbol.
- **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**. The main reason behind this restriction is that we can consider the different modules of a structure chart to be arranged in layers or levels. The principle of abstraction does not allow lower-level modules to be aware of the existence of the high-level modules. However, it is possible for t wo higher-level modules to invoke the same lower-levelmodule.
- An example of a properly layered design and another of a poorly layered design are shown in Figure6.18.

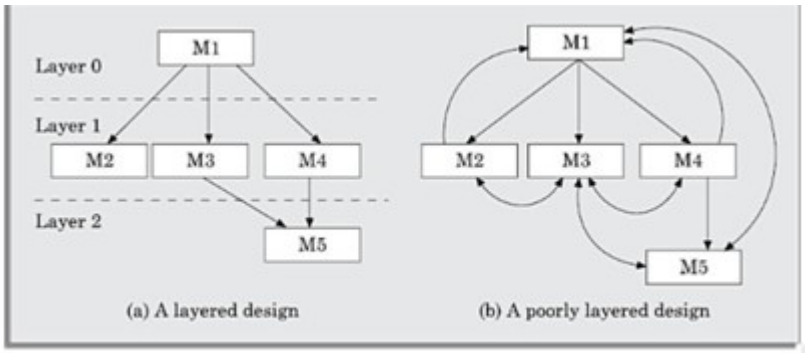


Figure 6.18: Examples of properly and poorly layered designs.

Figure 6.18: Examples of properly and poorly layered designs.

Flow chart *versus* structure chart

- Flow chart is a convenient technique to represent the flow of control in aprogram.
- A structure chart differs from a flow chart in three principalways:
    - It is usually difficult to identify the different modules of a program from its flow chartrepresentation.
    - Data interchange among different modules is not represented in a flowchart.
    - Sequential ordering of tasks that is inherent to a flow chart is suppressed in a structurechart.

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 by as a structure chart.
- Structured design provides two strategies to guide transformation of a DFD into a structurechart:
    Transform analysis
    Transaction analysis
- level 1 DFD, transform it into module representation using either the transform or transaction analysis and then proceed toward the lower levelDFDs.
- At each level of transformation, it is important to first determine whether the transform or the transaction analysis is applicable to a particularDFD.
- Whether to apply transform or transactionprocessing?
- Given a specific DFD of a model, how does one decide whether to apply transform analysis or transactionanalysis?
- For this, one would have to examine the data input to the diagram. The data input to the diagram can be easily spotted because they are represented by danglingarrows.
- 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. Normally, transform analysis is applicable only to very simpleprocessing.
- The bubbles are decomposed until it represents a very simple processing that can be implemented using only a few lines ofcode.
- Therefore, transform analysis is normally applicable at the lower levels of a DFD model. Each different way in which data is processed corresponds to a separate transaction.Each

transaction corresponds to a functionality that lets a user perform a meaningful piece of work using the software.

Transform analysis

- Transform analysis identifies the primary functional components (modules) and the input and output data for thesecomponents.
- The first step in transform analysis is to divide the DFD into three types ofparts:
  • Input.
  • Processing.
  • Output.
- The input portion in the DFD includes processes that transform input data from physical (e.g, character from terminal) to logical form (e.g. internal tables, lists,etc.).
- Each input portion is called an 'afferent branch' (not comparablebranch).
- The output portion of a DFD transforms output data from logical form to physicalform.
- Each output portion is called an efferent branch. The remaining portion of a DFD is called centraltransform.
- 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 thesemodules.
- Identifying the input and output parts requires experience and skill. One possible approach is to trace the input data until a bubble is found whose output data cannot be deduced from its inputs alone. Processes which validate input are not central transforms. Processes which sort input or filter data from it are centraltansforms.
- The first level o f structure chart is produced by representing each input and output unit as a box and each central transform as a singlebox.
- In the third step of transform analysis, the structure chart is refined by adding subfunctions required by each of the high-level functionalcomponents.
- Many levels of functional components may be added. This process of breaking functional components into subcomponents is called factoring. Factoring includes adding read and write modules, error-handling modules, initialization and termination process, identifying consumer modulesetc.
- The factoring process is continued until all bubbles in the DFD are represented in the structurechart.

Example-6 Draw the structure chart for the RMS software of Example-1.

- By observing the level 1 DFD of Figure6.8,

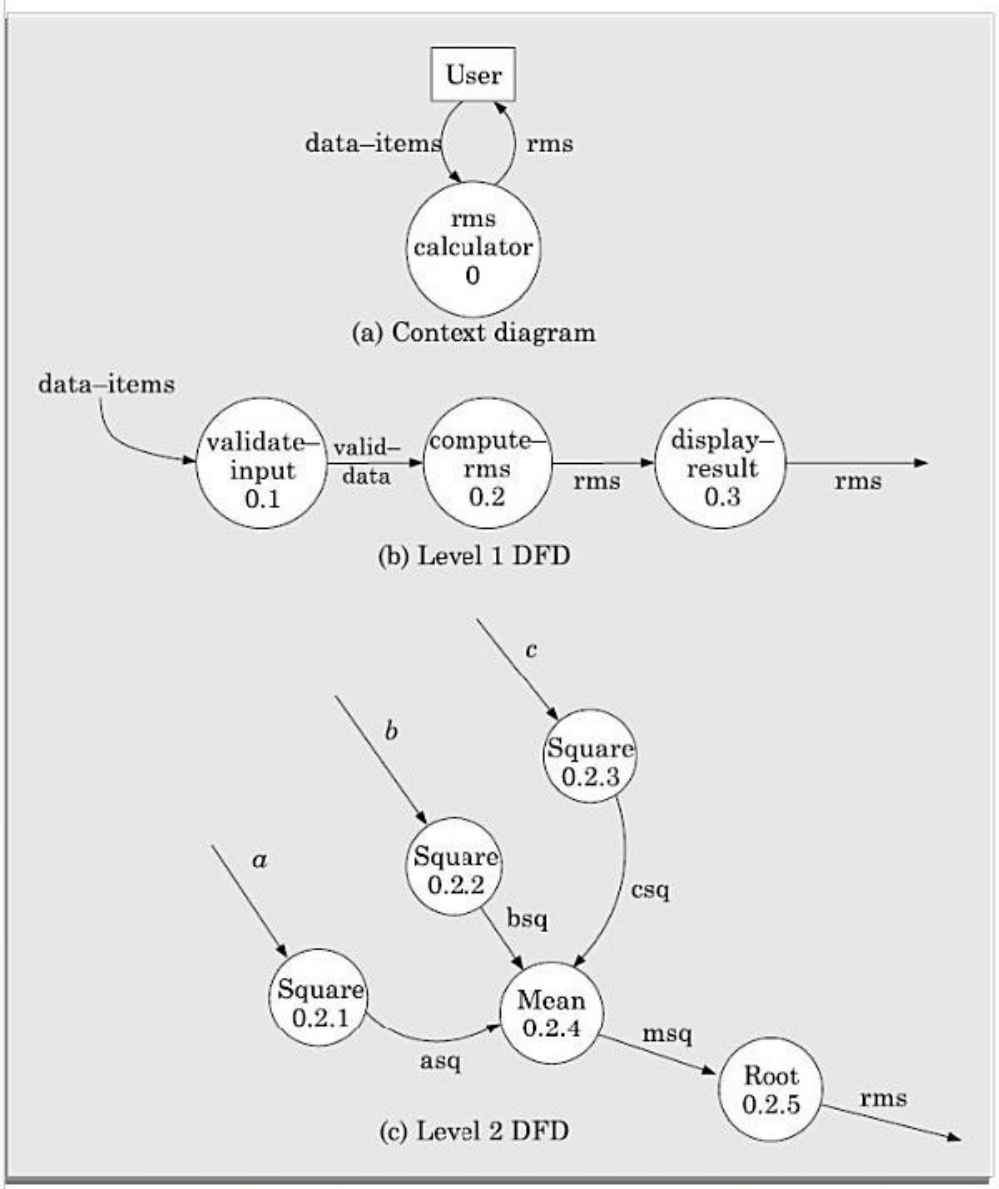


Figure 6.8: Context diagram, level 1, and level 2 DFDs for Example 6.1.

- we can identify validate-input as the afferent branch and write-output as the efferent branch. The remaining (i.e., compute-rms) as the central transform. By applying the step 2 and step 3 of transform analysis, we get the structure chart shown in Figure6.19.

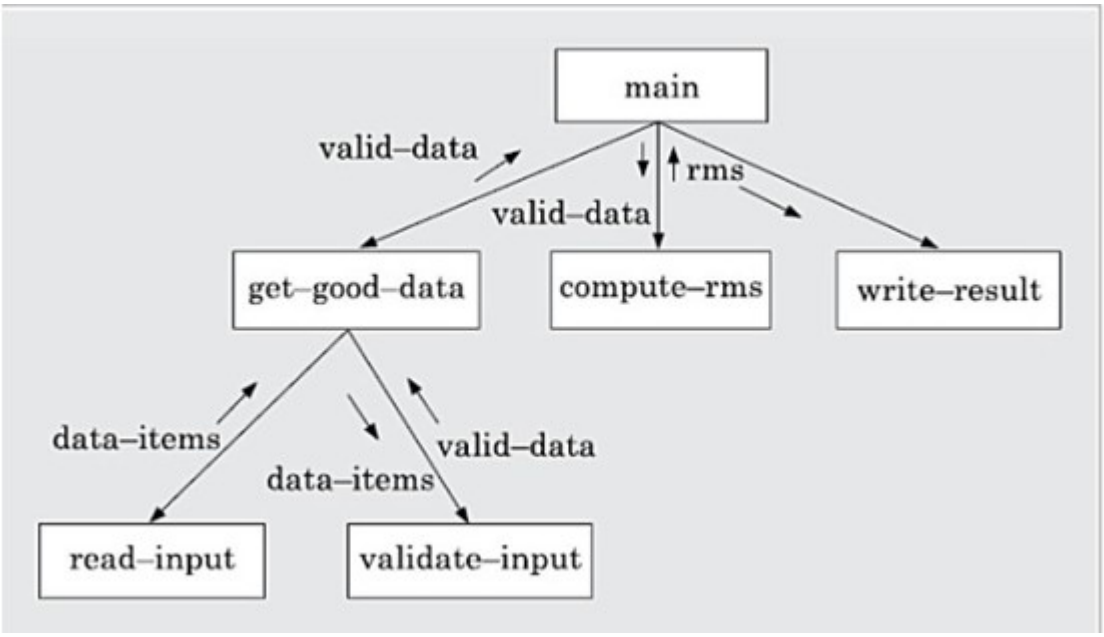


Figure 6.19: Structure chart for Example-1.

5. DETAILEDDESIGN

- During detailed design the pseudo code description of the processing and the different data structures are designed for the different modules of the structurechart.
- These are usually described in the form of module specifications (MSPEC). MSPEC is usually written using structuredEnglish.
- The MSPEC for the non-leaf modules describe the different conditions under which the responsibilities are delegated to the lower levelmodules.
- The MSPEC for the leaf-level modules should describe in algorithmic form how the primitive processing steps are carriedout.
- 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 themodule.

6. 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, who may or may not be the members of the developmentteam.
- Normally, members of the team who would code the design, and test the code, the analysts, and the maintainers attend the review meeting. The review team checks the design documents especially for the followingaspects:
- **Traceability:** Whether each bubble of the DFD can be traced to some module in the structure chart a nd vice versa. They check whether each functional requirement in the SRS document can be traced to some bubble in the DFD model and viceversa.
- **Correctness:** Whether all the algorithms and data structures of the detailed design are correct.
- **Maintainability:** Whether the design can be easily maintained infuture.
- **Implementation:** Whether the design can be easily and efficiently beimplemented.
- After the points raised by the reviewers is addressed by the designers,the

design document becomes ready for implementation.

7. USER INTERFACEDESIGN

- The user interface portion of a software product is responsible for all interactions with the user.
- Almost every software product has a userinterface.
- In the early days of computer, no software product had any userinterface.
- The computers those days were batch systems and no interactions with the users were supported.
- Now, we know that things are very different—almost every software product is highly interactive.
- The user interface part of a software product is responsible for all interactions with the end-user. Consequently, the user interface part of any software product is of direct concern to theend-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 userdissatisfaction.
- Users become particularly irritated when a system behaves in an unexpected ways, i.e., issued commands do not carry out actions according to the intuitive expectations of the user.
- Normally, when a user starts using a system, he builds a mental model of the system and expects the system behaviour to conform toit.
- For example, if a user action causes one type of system activity and response under some context, then the user would expect similar system activity and response to occur for similar user actions in similarcontexts.
- Therefore, sufficient care and attention should be paid to the design of the user interface of any softwareproduct.
- Development of a good user interface usually takes 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 increasetremendously.
- Therefore, it is necessary to carefully study various concepts associated with user interface design and understand various systematic techniques available for the development of userinterface.
- In this chapter, we first discuss some common terminologies and concepts associated with development of userinterfaces.
- Then, we classify the different types of interfaces commonly being used. We also provide some guidelines for designing good interfaces, and discuss some tools for development of graphical user interfaces (GUIs). Finally, we present a GUI developmentmethodology.

CHARACTERISTICS OF A GOOD USERINTERFACE

- It is important to identify the different characteristics that are usually desired of a good userinterface.
- Unless we know what exactly is expected of a good user interface, we cannot possibly designone.

- In the following subsections, we identify a few important characteristics of a good user interface:
- **Speed of learning:** A good user interface should be easy to learn. Speed of learning is hampered by complex syntax and semantics of the command issue procedures. **A good user interface should not require its users to memorise commands.** Neither should the user be asked to remember information from one screen to another while performing various tasks using the interface. Besides, the following three issues are crucial to enhance the speed oflearning:
    - **Use of metaphors and intuitive command names:** If the user interface of a text editor uses concepts similar to the tools used by a writer for text editing such as cutting lines and paragraphs and pasting it at other places, **users can immediately relate to it.** Another popular metaphor is a shopping cart. Everyone knows how a shopping cart is used to make choices while purchasing items in a supermarket. If a user interface uses the shopping cart metaphor for designing the interaction style for a situation where similar types of choices have to be made, then the users can easily understand and learn to use theinterface.
    - **Consistency:** Once, a user learns about a command, he should be able to use the similar commands in different circumstances for carrying out similar actions. This makes it easier to learn the interface since the user can extend his knowledge about one part of the interface to the other parts. Thus, the different commands supported by an interface should beconsistent.
    - **Component-based interface:** Users can learn an interface faster if the interaction style of the interface is very similar to the interface of other applications with which the user is already familiar with. This can be achieved if the interfaces of different applications are developed using some standard user interface components.
- **Speed of use:** Speed of use of a user interface is determined by the time and user effort necessary to initiate and execute differentcommands.
    - This characteristic of the interface is some times referred to as productivity support of the interface. It indicates how fast the users can perform their intended tasks.
    - The time and user effort necessary to initiate and execute different commands should be minimal. This can be achieved through careful design of the interface. For example, an interface that requires users to type in lengthy commands or involves mouse movements to different areas of the screen that are wide apart for issuing commands can slow down the operating speed ofusers.
    - The most frequently used commands should have the smallest length or be available at the top of a menu to minimise the mouse movements necessary to issue commands.
- **Speed of recall:** Once users learn how to use an interface, the speed with which they can recall the command issue procedure should be maximised. This characteristic is very important for intermittent users. Speed of recall isimproved if the interface is based on some metaphors, symbolic command issue procedures, and intuitive commandnames.
- **Error prevention:** A good user interface should minimise the scope of committing errors while initiating different commands. The error rate of an interface can beeasily

determined by monitoring the errors committed by an average users while using the interface.

- **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-basedinterfaces.
- **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 toanother.
- **Feedback:** A good user interface must provide feedback to various user actions. Especially, if any user request takes more than few seconds to process, the user should be informed about the state of the processing of hisrequest.
- **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 usabilityaspects.
- **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 ofcontrol.
- **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 andhelp.

8. BASICCONCEPTS

- **1. User Guidance and On-line Help :**Users may seek help about the operation of the software any time while using the software. This is provided by the on-line helpsystem.
    - **On-line help system:** Users expect the on-line help messages to be tailored to the context in which they invoke the "help system". Therefore, a good 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-dependentway.
    - **Guidance messages:** The guidance messages should be carefully designed to prompt the user about the next actions he might pursue, the current status of the system, the progress so far made in processing his last command, etc. A good guidance system should have different levels of sophistication for different categories of users. For example, a user using a command language interface might need a different type of guidance compared to a user using a menu or iconic interface
    - **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 someexceptional conditions, such as out of memory, communication link broken,etc.

- **2. Mode-based versus Modeless Interface:**
  - A **mode** is a state or collection of states in which only a subset of all user interaction tasks can beperformed.
  - a **modeless** interface, the same set of commands can be invoked at any time during the running of thesoftware.
  - Thus, a modeless interface has only a single mode and all the commands are available all the time during the operation of thesoftware.
  - On the other hand, in a mode-based interface, different sets of commands can be invoked depending on the mode in which the system is, i.e., the mode at any instant is determined by the sequence of commands already issued by theuser.
  - A mode-based interface can be represented using a state transition diagram, where each node of the state transition diagram would represent amode.
  - Each state of the state transition diagram can be annotated with the commands that are meaningful in thatstate.
- **3 Graphical User Interface (GUI) versus Text-based User Interface :**
  - In a **GUI** multiple windows with different information can simultaneously be displayed on the userscreen.
  - This is perhaps one of the biggest advantages of GUI over text- based interfaces since the user has the flexibility to simultaneously interact with several related items at anytime
  - Iconic information representation and symbolic information manipulation is possible in aGUI.
  - Symbolic information manipulation such as dragging an icon representing a file to a trash for deleting is intuitively very appealing and the user can instantly rememberit.
  - A GUI usually supports command selection using an attractive and user-friendly menu selectionsystem.
  - In a GUI, a **pointing device** such as a mouse or a **light pen** can be used for issuingcommands.
  - The use of a **pointing device** increases the efficacy of command issueprocedure.
  - A GUI requires special terminals with graphics capabilities for running and also requires special input devices such amouse.
  - **A text-based user interface** can be implemented even on a cheap alphanumeric display terminal. Graphics terminals are usually much more expensive than alphanumericterminals.
  - However, display terminals with graphics capability with bitmapped high-resolution displays and significant amount of local processing power have become affordable and over the years have replaced text-based terminals on alldesktops.

9. TYPES OF USERINTERFACES

- Broadly speaking, user interfaces can be classified into the following threecategories:

  Command language-based interfaces

  Menu-based interfaces

  Direct manipulation interfaces

- Each of these categories of interfaces has its own characteristic advantages and disadvantages.
- Therefore, most modern applications use a careful combination of all these three types of user interfaces for implementing the user commandrepertoire.
- It is very difficult to come up with a simple set of guidelines as to which parts of the interface should be implemented using what type ofinterface.
- This choice is to a large extent dependent on the experience and discretion of the designer of theinterface.

1. Command Language-basedInterface

- A command language-based interface—as the name itself suggests, is based on designing a command language which the user can use to issue thecommands.
- The user is expected to frame the appropriate commands in the language and type them appropriately wheneverrequired.
- A simple command language-based interface might simply assign unique names to the differentcommands.
- However, a more sophisticated command language-based interface may allow users to compose complex commands by using a set of primitivecommands.
- A command language-based interface can be made concise requiring minimal typing by the user.
- Command language-based interfaces allow fast interaction with the computer and simplify the input of complexcommands.
- Among the three categories of interfaces, the command language interface allows for most efficient command issue procedure requiring minimaltyping.
- Further, a command language-based interface can be implemented even on cheap alphanumeric terminals.
- Also, a command language-based interface is easier to develop compared to a menu-based or a direct-manipulation interface because compiler writing techniques are well developed.
- One can systematically develop a command language interface by using the standard compiler writing tools Lex andYacc.
- However, command language-based interfaces suffer from several drawbacks. Usually, command language-based interfaces are **difficult to learn** and require the user to **memorise the set of primitivecommands.**
- Also, most **users make errors while formulating commands in the command language** and also while typingthem.
- Further, in a command language-based interface, all interactions with the system is through a key-board and cannot take advantage of effective interaction devices such as a mouse.

- Obviously, for casual and inexperienced users, **command language-based interfaces are notsuitable**.

2. Menu-basedInterface

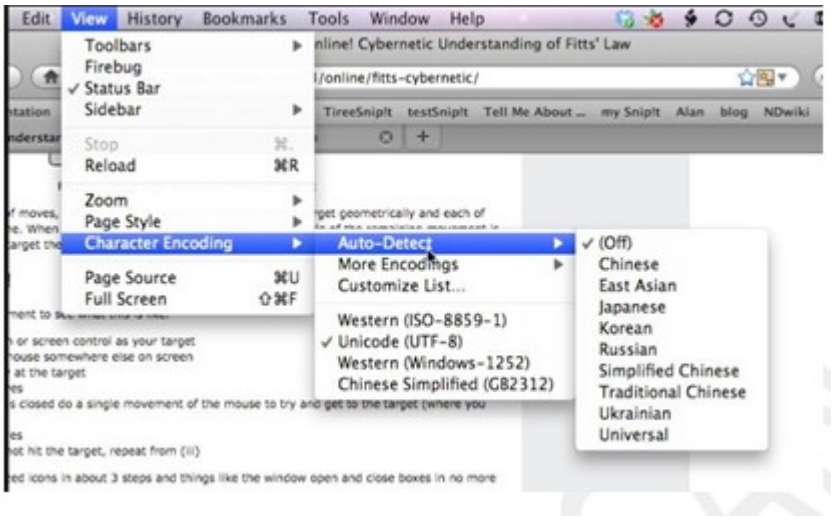
- An important advantage of a menu-based interface over a command language-based interface is that a menu-based interface does not require the users to remember the exact syntax of thecommands.
- A menu-based interface is based on recognition of the command names, rather than recollection.
- Humans are much better in recognizing something than recollectingit.
- Further, in a menu-based interface the typing effort is minimal as most interactions are carried out through menu selections using a pointingdevice.
- This factor is an important consideration for the occasional user who cannot typefast.
- However, experienced users find a menu-based user interface to be slower than a command language-based interface because an experienced user can type fast and can get speed advantage by composing different primitive commands to express complex commands.
- Composing commands in a menu based interface is notpossible.
- Also, if the number of choices is large, it is difficult to design a menu-basedinterfae.
- A moderate-sized software might need hundreds or thousands of different menu choices.
- In fact, a major challenge in the design of a menu-based interface is to structure large number of menu choices into manageableforms.
- In the following, the techniques available to structure a large number of menuitems:
- **Scrolling menu:** Sometimes the full choice list is large and cannot be displayed within the menu area, scrolling of the menu items isrequired.
    - This would enable the user to view and select the menu items that cannot be accommodated on thescreen.
    - In a scrolling menu all the commands should be highly correlated, so that the user can easily locate a command that heneeds.
    - This is important since the user cannot see all the commands at any onetime.
    - An example situation where a scrolling menu is frequently used is font size selection in a document processor (see Figure9.1).

Figure 9.1: Font size selection using scrolling menu.

- **Walking menu:** Walking menu is very commonly used to structure a large collection of menu items. In this technique, when a menu item is selected, it causes further menu items to be displayed adjacent to it in a sub-menu. An example of a walking menu is shown in Figure 9.2.

Figure 9.2: Example of walking menu.

- **Hierarchical menu:** This type of menu is suitable for small screens with limited display area such as that in mobile phones. In a hierarchical menu, the menu items are organised in a hierarchy or tree structure. Selecting a menu item causes the current menu display to be replaced by an appropriate sub-menu. Thus in this case, one can consider the menu and its various submenu to form a hierarchical tree-likestructure



3. Direct ManipulationInterfaces

- Direct manipulation interfaces present the interface to the user in the form of visual models (i.e., icons2 orobjects).
- For this reason, direct manipulation interfaces are sometimes called as iconic interfaces. In this type of interface, the user issues commands by performing actions on the visual representations of the objects, e.g., pull an icon representing a fileinto an icon representing a trash box, for deleting thefile.
- Important advantages of iconic interfaces include the fact that the icons can be recognised by the users very easily, and that icons are languageindependent.

10. FUNDAMENTALS OF COMPONENT-BASED GUI DEVELOPMENT

- Graphical user interfaces became popular in the1980s.
- The main reason why there were very few GUI-based applications prior to the eighties is that graphics terminals were tooexpensive.
- For example, the price of a graphics terminal those days was much more than what a high-end personal computer costs thesedays.
- The window system lets the application programmer create and manipulate windows without having to write the basic windowingfunctions.
- In the following subsections, an overview of the window management system, the component-based development style, and visualprogramming.

Window System

- Most modern graphical user interfaces are developed using some window system. A window system can generate displays through a set of windows. Since a window is the basic entity in such a graphical user interface, we need to first discuss what exactly a windowis.
- **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.

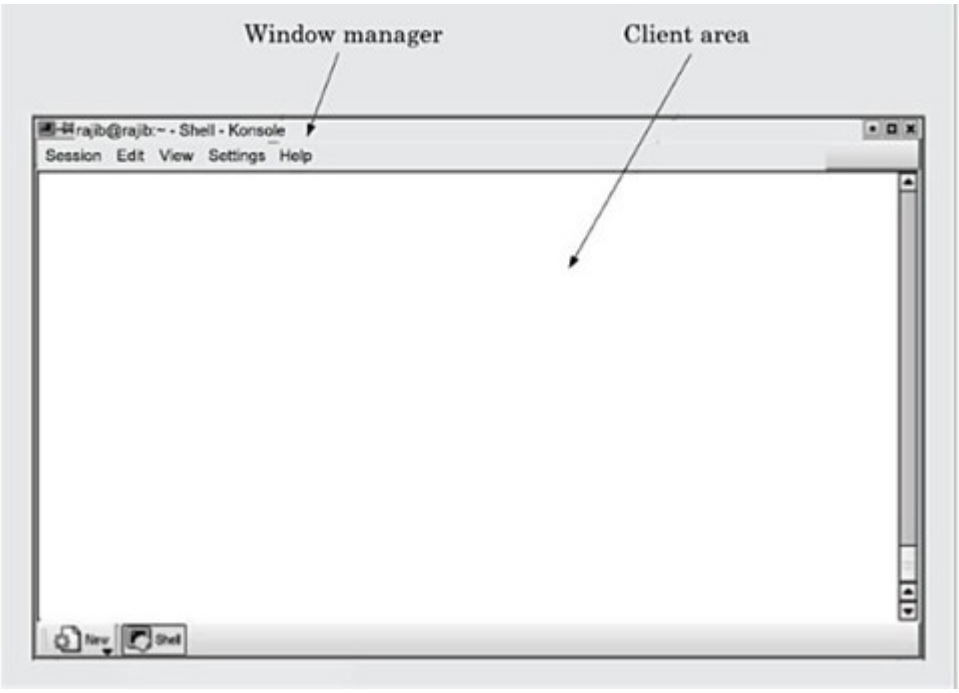


Figure 9.3: Window with client and user areas marked.

- A window can be divided into two parts—**client part**, and **non-clientpart**.
- The client area makes up the whole of the window, except for the borders and scrollbars.
- The client area is the area available to a client application fordisplay.
- The non-client-part of the window determines the look and feel of thewindow.
- The look and feel defines a basic behaviour for all windows, such as creating, moving, resizing, iconifying of thewindows.
- The window manager is responsible for managing and maintaining the non-client area of a window. A basic window with its different parts is shown in Figure9.3.

- **Window Management System (WMS) :** A graphical user interface typically consists of a large number ofwindows.
  - Therefore, it is necessary to have some systematic way to manage thesewindows.
  - Most graphical user interface development environments do this through a window management system (WMS).
  - A window management system is primarily a resourcemanager.
  - 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 behaviour 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 behaviour to the various windows such as move, resize, iconify, etc. A WMS consists of two parts (see Figure 9.4): a **window manager**, and a **window system**.

- **Window manager and window system:** 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 andbehave.
    - The window manager can be considered as a special kind of client that makes use of the services (function calls) supported by the windowsystem.
    - 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 Figure9.4.
    - 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 manger invoke services of the windowmanager.
    - 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.

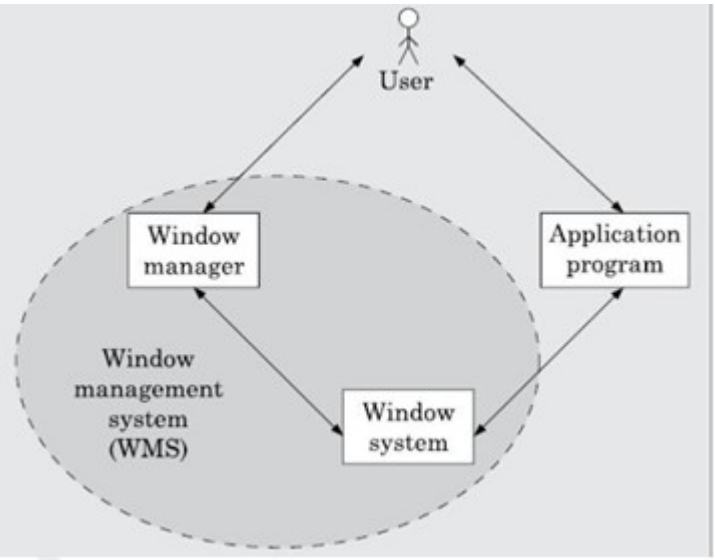


Figure 9.4: Window management system.

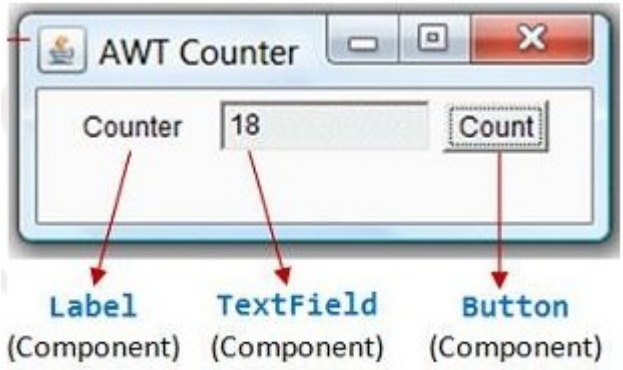
Component-baseddevelopment

- A development style based on widgets is called component-based (or widget-based) GUI developmentstyle.
- There are several important advantages of using a widget-based designstyle.
- One of the most important reasons to use widgets as building blocks is because they help users learn an interface fast. In this style of development, the user interfaces for different applications are built from the same basiccomponents.
- **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 programmingenvironment.
    - 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 whereverrequired.

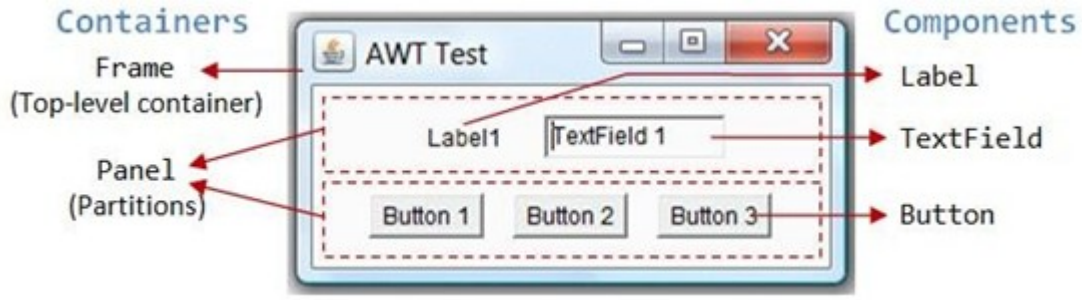
- Thus, visual programming can be considered as program development through manipulation of several visualobjects.
- Reuse of program components in the form of visual objects is an important aspect of this style ofprogramming.
- Though popular for user interface development, this style of programming 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 theinterface.
- 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 Windowsenvironments.

Types ofWidgets

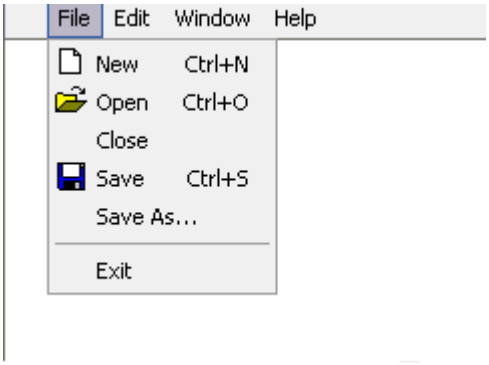
- Widget is an application, or a component of an interface, that enables a user to perform a function or access aservice.
- Different interface programming packages support different widgetsets.
- However, a surprising number of them contain similar kinds ofwidgets.
- The following widgets we have chosen as representatives of this genericclass.
- **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 otherwidgets.



- **Container widget:** These widgets do not stand by themselves, but exist merely tocontain 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 withit.



- **Pop-up menu:** These are transient and task specific. A pop-up menu appears upon pressing the mouse right button, irrespective of the mouseposition.
- **Pull-down menu :** These are more permanent and general. You have to move the cursor to a specific location and pull down this type ofmenu.



- **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. Though most dialog boxes ask you to enter someinformation, 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 willappear.
- **Radio buttons:** A set of radio buttons are 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 oldradios.
- **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.

An Overview ofX-Window/MOTIF

- 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 portableGUIs.**
- **Applications developed using the 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 Figure9.5.

- Here, A is the computer application in which the application is running. B can be any computer on the network from where you can interact with theapplication.
- 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 ofJava.

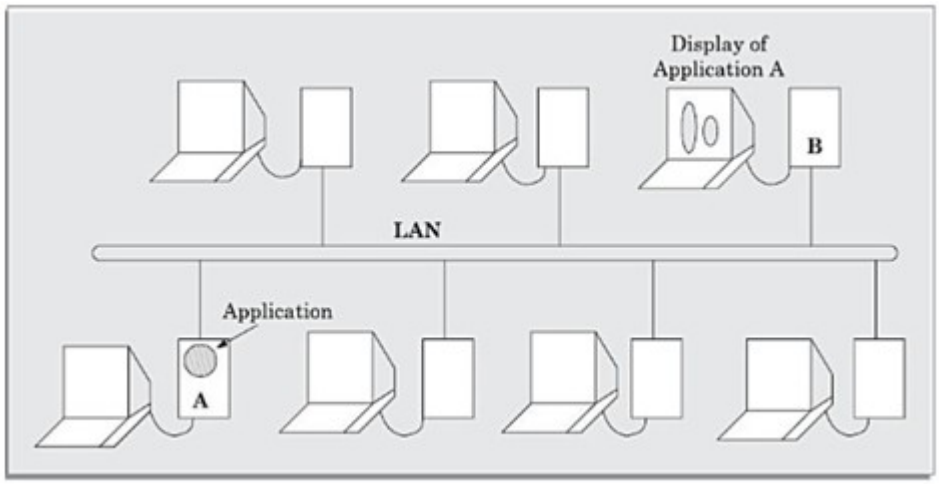


Figure 9.5: Network-independent GUI.

- 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 libraryroutines.

X Architecture

- The X architecture is pictorially depicted in Figure 9.6. The different terms used in this diagram are explained asfollows:

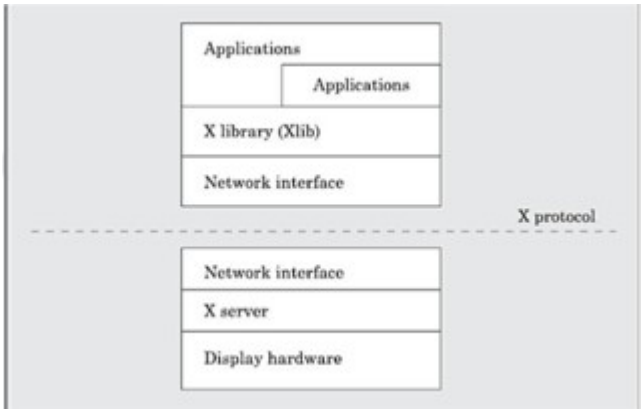


Figure 9.6: Architecture of the X System.

- **Xserver:** The X server runs on the hardware to which the display and the key board are 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 managesit.
- **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 characters 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, menu 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 behaviour 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 Xlibrary.

11. A USER INTERFACE DESIGNMETHODOLOGY

- At present, no step-by-step methodology is available which can be followed by rote to come up with a good userinterface.
- What we present in this section is a set of recommendations which you can use to complement your ingenuity. Even though almost all popular GUI design methodologies are user-centered, this concept has to be clearly distinguished from a user interface design byusers.
- Before we start discussing about the user interface design methodology, let us distinguish between a user-centered design and a design byusers.
- User-centered design is the theme of almost all modern user interface designtechniques.
- 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 alwaysright.

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. In the following subsections, we discuss some of the prominent issues that have been extensively reported in theliterature.
- **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 tosee.
- **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 t a sk closure.

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

A GUI DesignMethodology

- 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:
- Detailed presentation and graphics design. GUI construction. Usabilityevaluation.

Examining the use case model

- We now elaborate the above steps in GUI design. The starting point for GUI design is the use casemodel.
- 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 moremetaphors.
- **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 trainingcosts.
- Some commonly used metaphors are thefollowing:
  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 modelling

- A task is a human activity intended to achieve some goals. Examples of task goals can be asfollows:
  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 atask.
- A task model should show the structure of the subtasks that the user needs to perform to achieve the overall taskgoal.
- Each task can be modeled as a hierarchy ofsubtasks.
- 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 of decomposition of a task into subtasks is shown in Figure9.7.

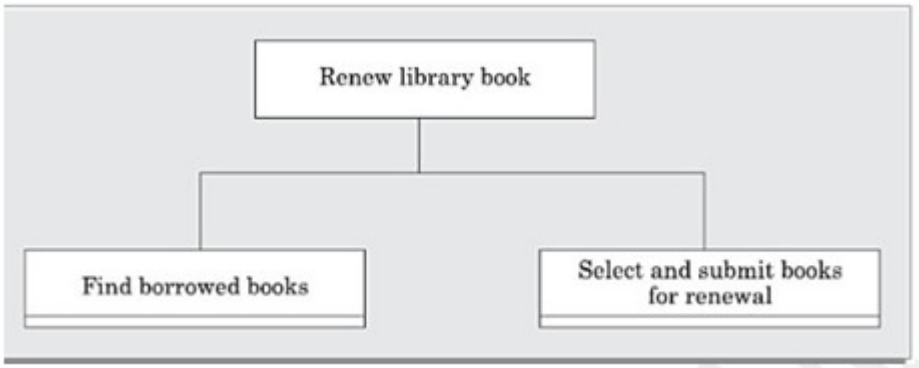


Figure 9.7: Decomposition of a task into subtasks.

- Identification of the user objects forms the basis of an object-baseddesign.
- A user object model is a model of business objects which the end-users believe that they are interactingwith.
- The objects in a library software may be books, journals, members,etc.
- The objects in the supermarket automation software may be items, bills, indents, shopping list,etc.
- The state diagram for an object can be drawn using a notation similar to that used by UML..
- The state diagram of an object model can be used to determine which menu items should be dimmed in astate.
- An example state chart diagram for an order object is shown in Figure9.8.

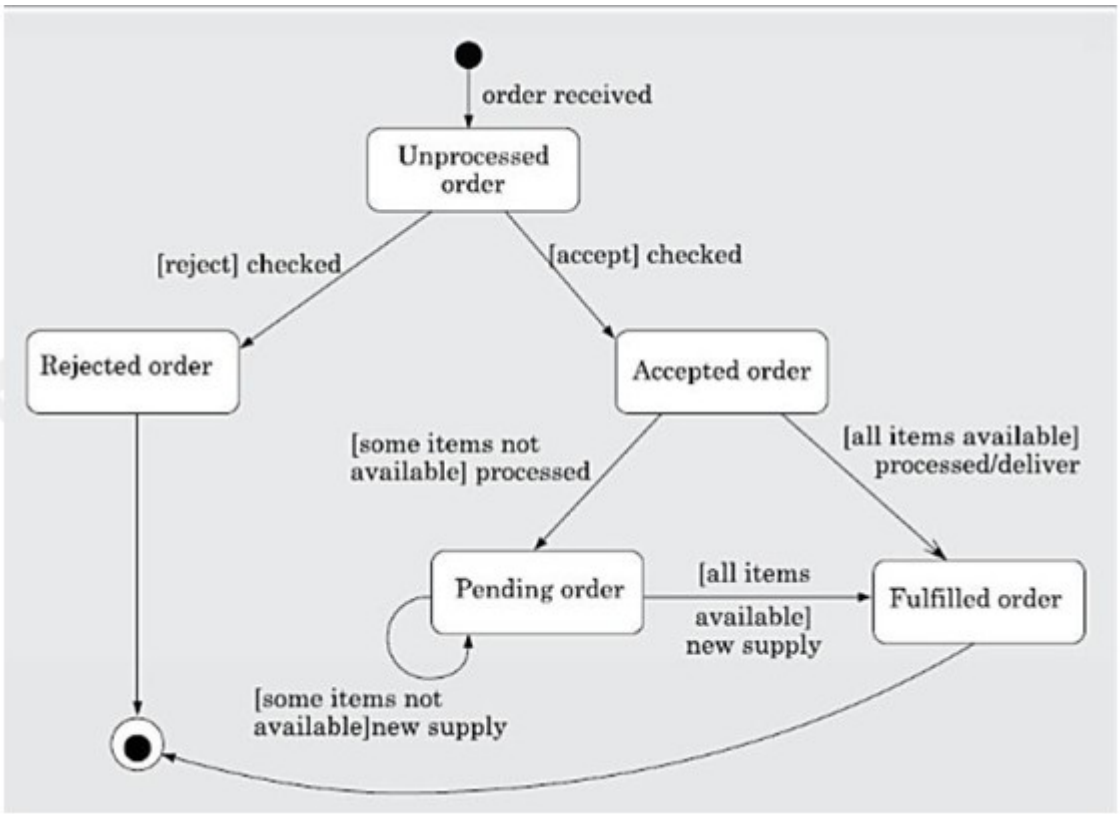


Figure 9.8: State chart diagram for an order object.

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 concreteobjects.
- **Example 9.1** We need to develop the interface for a web-based pay-order shop, where the users can examine the contents of the shop through a web browser and can order them. Several metaphors are possible for different parts of this problem as follows: 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 organised in the form of a book, similar to the way information abo u t electronic components are organised 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 shoppingcart.
- **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, an other widgets to be organised 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 differentobjects.
- **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 is 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 (e.g., confirmation of delete). Though use of modal dialogs are essential in some situations, overuse of modal dialogs reduces user flexibility. In particular, sequences of modal dialogs should beavoided.
- **User interface inspection : B**uild a check list of points which can be easily checked for aninterface.
- The check listdetails are asfollows.
- **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 goingon.
- **Match between the system and the real world:** The system should speak the user's language with words, phrases, and concepts familiar to that used by the user, rather than using system-orientedterms.
- **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 redooperations.

- **Consistency:** The users should not have to wonder whether different words, concepts, and operations mean the same thing in differentsituations.
- **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 theuser.
- **Support for multiple skill levels:** Provision of accelerators for experienced users allows them to efficiently carry out the actions they most frequently require toperform.
- **Aesthetic and minimalist design:** Dialogs and screens should not contain information which are irrelevant and are rarely needed. Every extra unit of information in a dialog or screen competes with the relevant units and diminishes theirvisibility.
- **Help and error messages:** These should be expressed in plain language (no codes), precisely indicating the problem, and constructively suggesting asolution.
- **Error prevention:** Error possibilities should be minimised. A key principle in this regard is to prevent the user from entering wrongvalues.