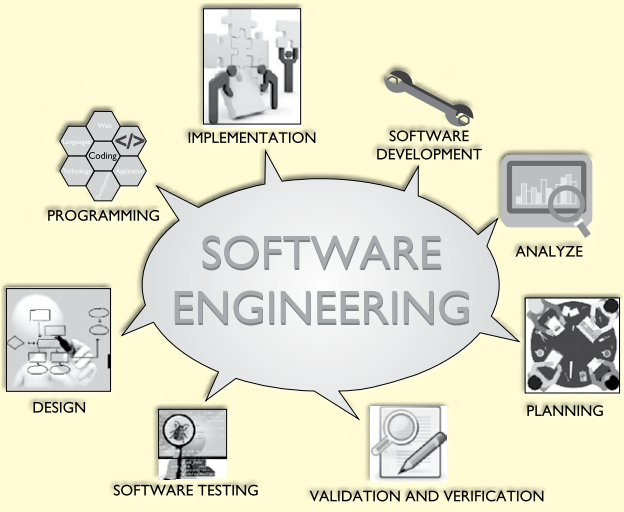
CHAPTER

5

SOFTWARE DESIGN



CHAPTER OUTLINE

- Overview of the Design Process
- How to Characterise a Good Software Design?
- Cohesion and Coupling
- Layered Arrangement of Modules
- Approaches to Software Design

Design can be simple. That's why it's so complicated.

—Paul Rand

LEARNING OBJECTIVES

- High-level versus detailed design
- Characteristics of a good design
- Determination of the levels of cohesion and coupling of a module
- Procedural versus object-oriented design

During the software design phase, the design document is produced, based on the customer requirements as documented in the SRS document.

This view of a design process has been shown schematically in Figure 5.1. As shown in Figure 5.1, the design process starts using the SRS document and completes with the production of the design document. The design document produced at the end of the design phase should be implementable using a programming language in the subsequent (coding) phase.

> The activities carried out during the design phase (called as *design process*) transform the SRS document into the design document.

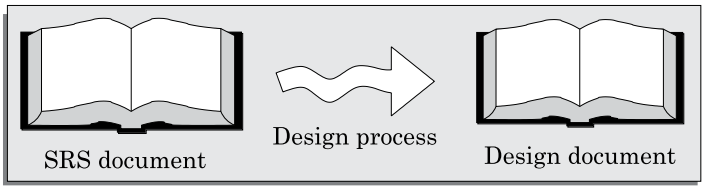


FIGURE 5.1 The design process.

5.1 OVERVIEW OF THE DESIGN PROCESS

The design process essentially transforms the SRS document into a design document. In the following sections and subsections, we will discuss a few important issues associated with the design process.

5.1.1 Outcome of the Design Process

The following items are designed and documented during the design phase.

Different modules required: The different modules in the solution should be identified. Each module is a collection of functions and the data shared by these functions. Each module should accomplish some well-defined task out of the overall responsibility of the software. Each module should be named according to the task it performs. For example, in an academic automation software, the module consisting of the functions and data necessary to accomplish the task of registration of the students should be named `handle student registration`.

Control relationships among modules: A control relationship between two modules essentially arises due to *function calls* across the two modules. The control relationships existing among various modules should be identified in the design document.

Interfaces among diffrent modules: The interfaces between two modules identifies the exact data items that are exchanged between the two modules when one module invokes a function of the other module.

Data structures of the individual modules: Each module normally stores some data that the functions of the module need to share to accomplish the overall responsibility of the module. Suitable data structures for storing and managing the data of a module need to be properly designed and documented.

Algorithms required to implement the individual modules: Each function in a module usually performs some processing activity. The algorithms required to accomplish the processing activities of various modules need to be carefully designed and documented with due considerations given to the accuracy of the results, space and time complexities.

Starting with the SRS document (as shown in Figure 5.1), the design documents are produced through iterations over a series of steps that we are going to discuss in this chapter and the subsequent three chapters. The design documents are reviewed by the members of the development team to ensure that the design solution conforms to the requirements specification.

5.1.2 Classification of Design Activities

A good software design is seldom realised by using a single step procedure, rather it requires iterating over a series of steps called the *design activities*. Let us first classify the design activities before discussing them in detail. Depending on the order in which various design activities are performed, we can broadly classify them into two important stages.

- Preliminary (or high-level) design, and
- Detailed design.

Through high-level design, a problem is decomposed into a set of modules. The control relationships among the modules are identified, and also the interfaces among various modules are identified.

The meaning and scope of these two stages can vary considerably from one design methodology to another. However, for the traditional function-oriented design approach, it is possible to define the objectives of the high-level design as follows:

The outcome of high-level design is called the *program structure* or the *software architecture.* High-level design is a crucial step in the overall design of a software. When the high-level design is complete, the problem should have been decomposed into many small functionally independent modules that are cohesive, have low coupling among themselves, and are arranged in a hierarchy. Many different types of notations have been used to represent a high-level design. A notation that is widely being used for procedural development is a tree-like diagram called the *structure chart*. Another popular design representation techniques called UML that is being used to document object-oriented design, involves developing several types of diagrams to document the object-oriented design of a systems. Though other notations such as Jackson diagram [1975] or

Warnier-Orr [1977, 1981] diagram are available to document a software design, we confine our attention in this text to structure charts and UML diagrams only.

Once the high-level design is complete, detailed design is undertaken.

During detailed design each module is examined carefully to design its data structures and the algorithms.

The outcome of the detailed design stage is usually documented in the form of a *module specification* (MSPEC) document. After the high-level design is complete, the problem would have been decomposed into small modules, and the data structures and algorithms to be used described using MSPEC and can be easily grasped by programmers for initiating coding. In this text, we do not discuss MSPECs and confine our attention to high-level design only.

5.1.3 Classification of Design Methodologies

The design activities vary considerably based on the specific design methodology being used. A large number of software design methodologies are available. We can roughly classify these methodologies into procedural and object-oriented approaches. These two approaches are two fundamentally different design paradigms. In this chapter, we shall discuss the important characteristics of these two fundamental design approaches. Over the next three chapters, we shall study these two approaches in detail.

Do design techniques result in unique solutions?

Even while using the same design methodology, different designers usually arrive at very different design solutions. The reason is that a design technique often requires the designer to make many subjective decisions and work out compromises to contradictory objectives. As a result, it is possible that even the same designer can work out many different solutions to the same problem. Therefore, obtaining a good design would involve trying out several alternatives (or candidate solutions) and picking out the best one. However, a fundamental question that arises at this point is—how to distinguish superior design solution from an inferior one? Unless we know what a good software design is and how to distinguish a superior design solution from an inferior one, we can not possibly design one. We investigate this issue in the next section.

Analysis *versus* design

The goal of any analysis technique is to elaborate the customer requirements through careful thinking and at the same time consciously avoiding making any decisions regarding the exact way the system is to be implemented.

Analysis and design activities differ in goal and scope.

The analysis results are generic and does not consider implementation or the issues associated with specific platforms. The analysis model is usually documented using some graphical formalism. In case of the function-oriented approach that we are going to discuss, the analysis model would be documented using *data flow diagrams* (DFDs), whereas the design would be documented using structure chart. On the other hand, for object-oriented approach, both the design model and the analysis model will be documented using *unified modelling language* (UML). The analysis model would normally be very difficult to implement using a programming language.

The design model is obtained from the analysis model through transformations over a series of steps. In contrast to the analysis model, the design model reflects several decisions taken regarding the exact way system is to be implemented. The design model should be detailed enough to be easily implementable using a programming language.

5.2 HOW TO CHARACTERISE A GOOD SOFTWARE DESIGN?

Coming up with an accurate characterisation of a good software design that would hold across diverse problem domains is certainly not easy. In fact, the definition of a "good" software design can vary depending on the exact application being designed. For example, "memory size used up by a program" may be an important way to characterize a good solution for embedded software development—since embedded applications are often required to work under severely limited memory sizes due to cost, space, or power consumption considerations. For embedded applications, factors such as design comprehensibility may take a back seat while judging the goodness of design. Thus for embedded applications, one may sacrifice design comprehensibility to achieve code compactness. Similarly, it is not usually true that a criterion that is crucial for some application, needs to be almost completely ignored for another application. It is therefore clear that the criteria used to judge a design solution can vary widely across different types of applications. Not only do the criteria used to judge a design solution depend on the exact application being designed, but to make the matter worse, there is no general agreement among software engineers and researchers on the exact criteria to use for judging a design even for a specific category of application. However, most researchers and software engineers agree on a few desirable characteristics that every good software design for general applications must possess. These characteristics are listed below:

Correctness: A good design should first of all be correct. That is, it should correctly implement all the functionalities of the system.

Understandability: A good design should be easily understandable. Unless a design solution is easily understandable, it would be difficult to implement and maintain it.

Efficiency: A good design solution should adequately address resource, time, and cost optimisation issues.

Maintainability: A good design should be easy to change. This is an important requirement, since change requests usually keep coming from the customer even after product release.

5.2.1 Understandability of a Design: A Major Concern

While performing the design of a certain problem, assume that we have arrived at a large number of design solutions and need to choose the best one. Obviously all incorrect designs have to be discarded first. Out of the correct design solutions, how can we identify the best one?

Recollect from our discussions in Chapter 1 that a good design should help overcome the human cognitive limitations that arise due to limited short-term memory. A large problem overwhelms the human mind, and a poor

Given that we are choosing from only correct design solutions, understandability of a design solution is possibly the most important issue to be considered while judging the goodness of a design.

design would make the matter worse. Unless a design solution is easily understandable, it could lead to an implementation having a large number of defects and at the same time tremendously pushing up the development costs. Therefore, a good design solution should be simple and easily understandable. A design that is easy to understand is also easy to develop and maintain. A complex design would lead to severely increased life cycle costs. Unless a design is easily understandable, it would require tremendous effort to implement, test, debug, and maintain it. We had already pointed out in Chapter 2 that about 60 per cent of the total effort in the life cycle of a typical product is spent on maintenance. If the software is not easy to understand, not only would it lead to increased development costs, the effort required to maintain the product would also increase manifold. Besides, a design solution that is difficult to understand would lead to a program that is full of bugs and is unreliable. Recollect that we had already discussed in Chapter 1 that understandability of a design solution can be enhanced through clever applications of the principles of abstraction and decomposition.

An understandable design is modular and layered

How can the understandability of two different designs be compared, so that we can pick the better one? To be able to compare the understandability of two design solutions, we should at least have an understanding of the general features that an easily understandable design should possess. A design solution should have the following characteristics to be easily understandable:

- It should assign consistent and meaningful names to various design components.
- It should make use of the principles of decomposition and abstraction in good measures to simplify the design.

We had discussed the essential concepts behind the principles of abstraction and decomposition principles in Chapter 1. But, how can the abstraction and decomposition principles are used in arriving at a design solution? These two principles are exploited by design methodologies to make a design modular and layered. (Though there are also a few other forms in which the abstraction and decomposition principles can be used in the design solution, we discuss those later). We can now define the characteristics of an easily understandable design as follows: A design solution is understandable, if it is modular and the modules are arranged in distinct layers.

We now elaborate the concepts of modularity and layering of modules:

A design solution should be modular and layered to be understandable.

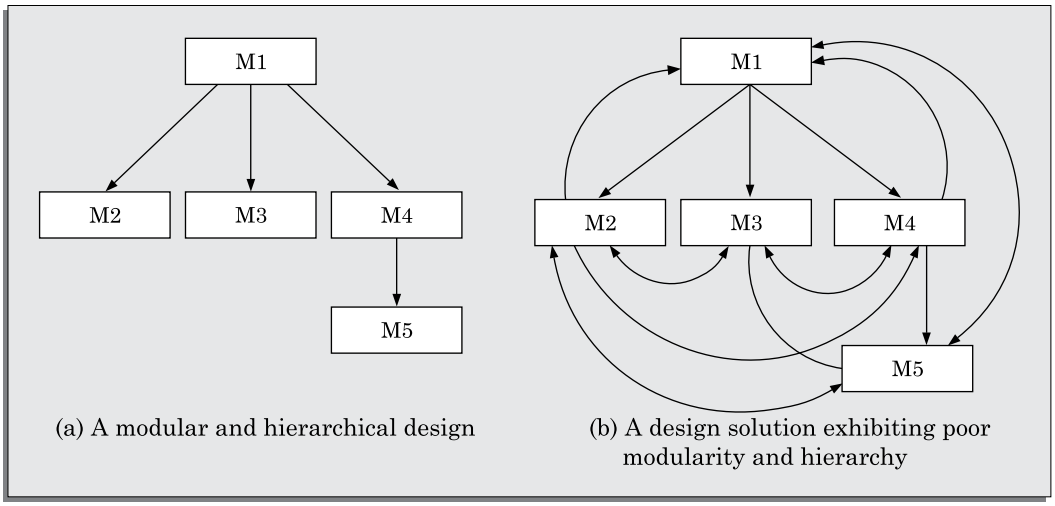
Modularity

A modular design is an effective decomposition of a problem. It is a basic characteristic of any good design solution. A *modular design*, in simple words, implies that the problem has been decomposed into a set of modules that have only limited interactions with each other. Decomposition of a problem into modules facilitates taking advantage of the *divide and conquer* principle. If different modules have either no interactions or little interactions with each other, then each module can be understood separately. This reduces the perceived complexity of the design solution greatly. To understand why this is so, remember that it may be very difficult to break a bunch of sticks which have been tied together, but very easy to break the sticks individually.

It is not difficult to argue that modularity is an important characteristic of a good design solution. But, even with this, how can we compare the modularity of two alternate design solutions? From an inspection of the module structure, it is at least possible to intuitively form an idea as to which design is more modular. For example, consider two alternate design solutions to a problem that are represented in Figure 5.2, in which the modules M_1, M_2, etc. have been drawn as rectangles. The invocation of a module by another module has been shown as an arrow. It can easily be seen that the design solution of Figure 5.2(a) would be easier to understand since the interactions among the different modules is low. But, can we quantitatively measure the modularity of a design solution? Unless we are able to quantitatively measure the modularity of a design solution, it will be hard to say which design solution is more modular than another. Unfortunately, there are no quantitative metrics available yet to directly measure the modularity of a design. However, we can quantitatively characterise the modularity of a design solution based on the cohesion and coupling existing in the design.

> A design solution is said to be highly modular, if the different modules in the solution have high cohesion and their inter-module couplings are low.

A software design with high cohesion and low coupling among modules is the effective problem decomposition we discussed in Chapter 1. Such a design would lead to increased productivity during program development by bringing down the perceived problem complexity.



(a) A modular and hierarchical design

(b) A design solution exhibiting poor modularity and hierarchy

FIGURE 5.2 Two design solutions to the same problem.

Based on this classification, we would be able to easily judge the cohesion and coupling existing in a design solution. From a knowledge of the cohesion and coupling in a design, we can form our own opinion about the modularity of the design solution. We shall define the concepts of cohesion and coupling and the various classes of cohesion and coupling in Section 5.3. Let us now discuss the other important characteristic of a good design solution—layered design.

Layered design

A layered design is one in which when the call relations among different modules are represented graphically, it would result in a tree-like diagram with clear layering. In a layered design solution, the modules are arranged in a hierarchy of layers. A module can only invoke functions of the modules in the layer immediately below it. The higher layer modules can be considered to be similar to managers that invoke (order) the lower layer modules to get certain tasks done. A layered design can be considered to be implementing *control abstraction*, since a module at a lower layer is unaware of (about how to call) the higher layer modules.

When a failure is detected while executing a module, it is obvious that the modules below it can possibly be the source of the error. This greatly simplifies debugging since one would need to concentrate only on a few modules to detect the error. We shall elaborate these concepts governing layered design of modules in Section 5.4.

A layered design can make the design solution easily understandable, since to understand the working of a module, one would at best have to understand how the immediately lower layer modules work without having to worry about the functioning of the upper layer modules.

5.3 COHESION AND COUPLING

We have so far discussed that effective problem decomposition is an important characteristic of a good design. Good module decomposition is indicated through *high cohesion* of the individual modules and *low coupling* of the modules with each other. Let us now define what is meant by cohesion and coupling.

In this section, we first elaborate the concepts of cohesion and coupling. Subsequently, we discuss the classification of cohesion and coupling.

Cohesion is a measure of the functional strength of a module, whereas the coupling between two modules is a measure of the degree of interaction (or interdependence) between the two modules.

Coupling: Intuitively, we can think of coupling as follows. Two modules are said to be highly coupled, if either of the following two situations arise:

- If the function calls between two modules involve passing large chunks of shared data, the modules are tightly coupled.
- If the interactions occur through some shared data, then also we say that they are highly coupled.

If two modules either do not interact with each other at all or at best interact by passing no data or only a few primitive data items, they are said to have low coupling.

Cohesion: To understand cohesion, let us first understand an analogy. Suppose you listened to a talk by some speaker. You would call the speech to be cohesive, if all the sentences of the speech played some role in giving the talk a single and focused theme. Now, we can extend this to a module in a design solution. When the functions of the module co-operate with each other for performing a single objective, then the module has good cohesion. If the functions of the module do very different things and do not co-operate with each other to perform a single piece of work, then the module has very poor cohesion.

Functional independence

By the term *functional independence,* we mean that a module performs a single task and needs very little interaction with other modules.

Functional independence is a key to any good design primarily due to the following advantages it offers:

A module that is highly cohesive and also has low coupling with other modules is said to be *functionally independent* of the other modules.

Error isolation: Whenever an error exists in a module, functional independence reduces the chances of the error propagating to the other modules. The reason behind this is that if a module is functionally independent, its interaction with other modules is low. Therefore, an error existing in the module is very unlikely to affect the functioning of other modules.

Further, once a failure is detected, error isolation makes it very easy to locate the error. On the other hand, when a module is not functionally independent, once a failure is detected in a functionality provided by the module, the error can be potentially in any of the large number of modules and propagated to the functioning of the module.

Scope of reuse: Reuse of a module for the development of other applications becomes easier. The reasons for this is as follows. A functionally independent module performs some well-defined and precise task and the interfaces of the module with other modules are very few and simple. A functionally independent module can therefore be easily taken out and reused in a different program. On the other hand, if a module interacts with several other modules or the functions of a module perform very different tasks, then it would be difficult to reuse it. This is especially so, if the module accesses the data (or code) internal to other modules.

Understandability: When modules are functionally independent, complexity of the design is greatly reduced. This is because of the fact that different modules can be understood in isolation, since the modules are independent of each other. We have already pointed out in Section 5.2 that understandability is a major advantage of a modular design. Besides the three we have listed here, there are many other advantages of a modular design as well. We shall not list those here, and leave it as an assignment to the reader to identify them.

5.3.1 Classification of Cohesiveness

Cohesiveness of a module is the degree to which the different functions of the module co-operate to work towards a single objective. The different modules of a design can possess different degrees of freedom. However, the different classes of cohesion that modules can possess are depicted in Figure 5.3. The cohesiveness increases from coincidental to functional cohesion. That is, coincidental is the worst type of cohesion and functional is the best cohesion possible. These different classes of cohesion are elaborated below.

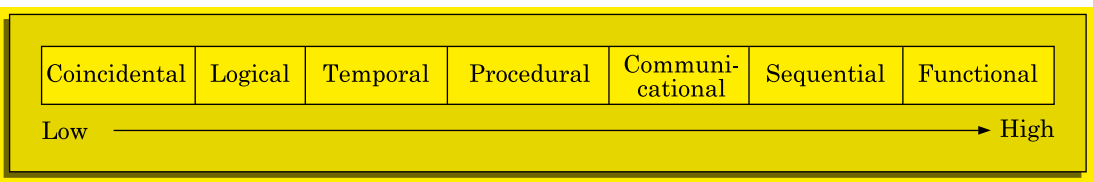


FIGURE 5.3 Classification of cohesion.

Coincidental cohesion: A module is said to have coincidental cohesion, if it performs a set of tasks that relate to each other very loosely, if at all. In this case, we can say that the module contains a random collection of functions. It is likely that the functions have been placed in the module out of pure coincidence rather than through some thought or design. The designs made by novice programmers often possess this category of cohesion, since they often bundle functions to modules rather arbitrarily. An example of a module with coincidental cohesion has been shown in Figure 5.4(a). Observe that the different functions of the module carry out very different and unrelated activities starting from issuing of library books to creating library member records on one hand, and handling librarian leave request on the other.

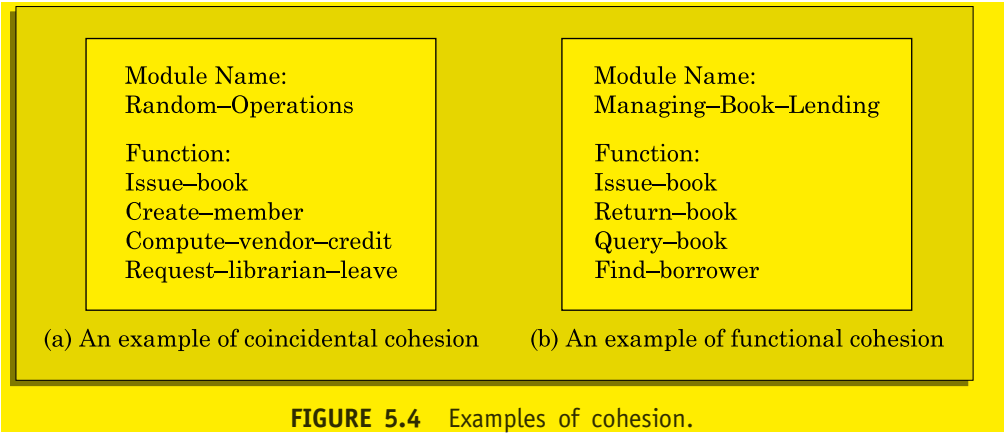


FIGURE 5.4 Examples of cohesion.

Logical cohesion: A module is said to be logically cohesive, if all elements of the module perform similar operations, such as error handling, data input, data output, etc. As an example of logical cohesion, consider a module that contains a set of print functions to generate various types of output reports such as grade sheets, salary slips, annual reports, etc.

Temporal cohesion: When a module contains functions that are related by the fact that these functions are executed in the same time span, then the module is said to possess temporal cohesion. As an example, consider the following situation. When a computer is booted, several functions need to be performed. These include initialisation of memory and devices, loading the operating system, etc. When a single module performs all these tasks, then the module can be said to exhibit temporal cohesion. Other examples of modules having temporal cohesion are the following. Similarly, a module would exhibit temporal cohesion, if it comprises functions for performing initialisation, or start-up, or shut-down of some process.

Procedural cohesion: A module is said to possess procedural cohesion, if the set of functions of the module are executed one after the other, though these functions may work towards entirely different purposes and operate on very different data. Consider the activities associated with order processing in a trading house. The functions `login()`, `place-order()`, `check-order()`, `print-bill()`, `place-order-on-vendor()`, `update-inventory()`, and `logout()` all do different thing and operate on different data. However,

they are normally executed one after the other during typical order processing by a sales clerk.

Communicational cohesion: A module is said to have communicational cohesion, if all functions of the module refer to or update the same data structure. As an example of procedural cohesion, consider a module named student in which the different functions in the module such as `admitStudent`, `enterMarks`, `printGradeSheet`, etc. access and manipulate data stored in an array named `studentRecords` defined within the module.

Sequential cohesion: A module is said to possess sequential cohesion, if the different functions of the module execute in a sequence, and the output from one function is input to the next in the sequence. As an example consider the following situation. In an on-line store consider that after a customer requests for some item, it is first determined if the item is in stock. In this case, if the functions `create-order()`, `check-item-availability()`, `place-order-on-vendor()` are placed in a single module, then the module would exhibit sequential cohesion. Observe that the function `create-order()` creates an order that is processed by the function `check-item-availability()` (whether the items are available in the required quantities in the inventory) is input to `place-order-on-vendor()`.

Functional cohesion: A module is said to possess functional cohesion, if different functions of the module co-operate to complete a single task. For example, a module containing all the functions required to manage employees' pay-roll displays functional cohesion. In this case, all the functions of the module (e.g., `computeOvertime()`, `computeWorkHours()`, `computeDeductions()`, etc.) work together to generate the payslips of the employees. Another example of a module possessing functional cohesion has been shown in Figure 5.4(b). In this example, the functions `issue-book()`, `return-book()`, `query-book()`, and `find-borrower()`, together manage all activities concerned with book lending. When a module possesses functional cohesion, then we should be able to describe what the module does using only one simple sentence. For example, for the module of Figure 5.4(a), we can describe the overall responsibility of the module by saying "It manages the book lending procedure of the library."

A simple way to determine the cohesiveness of any given module is as follows. First examine what do the functions of the module perform. Then, try to write down a sentence to describe the overall work performed by the module. If you need a compound sentence to describe the functionality of the module, then it has sequential or communicational cohesion. If you need words such as "first", "next", "after", "then", etc., then it possesses sequential or temporal cohesion. If it needs words such as "initialise", "setup", "shut down", etc., to define its functionality, then it has temporal cohesion.

We can now make the following observation. A cohesive module is one in which the functions interact among themselves heavily to achieve a single goal. As a result, if any of these functions is removed to a different module, the coupling would increase as the functions would now interact across two different modules.

5.3.2 Classification of Coupling

The degree of coupling between two modules depends on their interface complexity.

The coupling between two modules indicates the degree of interdependence between them. Intuitively, if two modules

interchange large amounts of data, then they are highly interdependent or coupled. We can alternately state this concept as follows.

The interface complexity is determined based on the number of parameters and the complexity of the parameters that are interchanged while one module invokes the functions of the other module.

Let us now classify the different types of coupling that can exist between two modules. Between any two interacting modules, any of the following five different types of coupling can exist. These different types of coupling, in increasing order of their severities have also been shown in Figure 5.5.

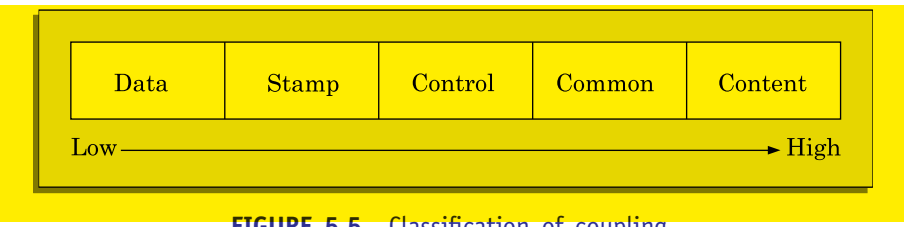


FIGURE 5.5 Classification of coupling.

Data coupling: Two modules are data coupled, if they communicate using an elementary data item that is passed as a parameter between the two, e.g. an integer, a float, a character, etc. This data item should be problem related and not used for control purposes.

Stamp coupling: Two modules are stamp coupled, if they communicate using a composite data item such as a record in PASCAL or a structure in C.

Control coupling: Control coupling exists between two modules, if data from one module is used to direct the order of instruction execution in another. An example of control coupling is a flag set in one module and tested in another module.

Common coupling: Two modules are common coupled, if they share some global data items.

Content coupling: Content coupling exists between two modules, if they share code. That is, a jump from one module into the code of another module can occur. Modern high-level programming languages such as C do not support such jumps across modules.

The different types of coupling are shown schematically in Figure 5.5. The degree of coupling increases from data coupling to content coupling. High coupling among modules not only makes a design solution difficult to understand and maintain, but it also increases development effort and also makes it very difficult to get these modules developed independently by different team members.

5.4 LAYERED ARRANGEMENT OF MODULES

The *control hierarchy* represents the organisation of program components in terms of their call relationships. Thus we can say that the control hierarchy of a design is determined by the order in which different modules call each other. Many different types of notations have been used to represent the control hierarchy. The most common notation is a tree-like diagram known as a *structure chart* which we shall study in some detail in Chapter 6.

However, other notations such as Warnier-Orr [1977, 1981] or Jackson diagrams [1975] may also be used. Since, Warnier-Orr and Jackson's notations are not widely used nowadays, we shall discuss only structure charts in this text.

In a layered design solution, the modules are arranged into several layers based on their call relationships. A module is allowed to call only the modules that are at a lower layer. That is, a module should not call a module that is either at a higher layer or even in the same layer. Figure 5.6(a) shows a layered design, whereas Figure 5.6(b) shows a design that is not layered. Observe that the design solution shown in Figure 5.6(b), is actually not layered since all the modules can be considered to be in the same layer. In the following, we state the significance of a layered design and subsequently we explain it.

An important characteristic feature of a good design solution is layering of the modules. A layered design achieves control abstraction and is easier to understand and debug.

In a layered design, the top-most module in the hierarchy can be considered as a manager that only invokes the services of the lower level module to discharge its responsibility. The modules at the intermediate layers offer services to their higher layer by invoking the services of the lower layer modules and also by doing some work themselves to a limited extent. The modules at the lowest layer are the worker modules. These do not invoke services of any module and entirely carry out their responsibilities by themselves.

Understanding a layered design is easier since to understand one module, one would have to at best consider the modules at the lower layers (that is, the modules whose services it invokes). Besides, in a layered design errors are isolated, since an error in one module can affect only the higher layer modules. As a result, in case of any failure of a module, only the modules at the lower levels need to be investigated for the possible error. Thus, debugging time reduces significantly in a layered design. On the other hand, if the different modules call each other arbitrarily, then this situation would correspond to modules arranged in a single layer. Locating an error would be both difficult and time consuming. This is because, once a failure is observed, the cause of failure (i.e. error) can potentially be in any module, and all modules would have to be investigated for the error. In the following, we discuss some important concepts and terminologies associated with a layered design:

Superordinate and subordinate modules: In a control hierarchy, a module that controls another module is said to be *superordinate* to it. Conversely, a module controlled by another module is said to be *subordinate* to the controller.

Visibility: A module B is said to be visible to another module A, if A directly calls B. Thus, only the immediately lower layer modules are said to be visible to a module.

Control abstraction: In a layered design, a module should only invoke the functions of the modules that are in the layer immediately below it. In other words, the modules at the higher layers, should not be visible (that is, abstracted out) to the modules at the lower layers. This is referred to as *control abstraction.*

Depth and width: Depth and width of a control hierarchy provide an indication of the number of layers and the overall span of control respectively. For the design of Figure 5.6(a), the depth is 3 and width is also 3.

Fan-out: Fan-out is a measure of the number of modules that are directly controlled by a given module. In Figure 5.6(a), the fan-out of the module M1 is 3. A design in which the modules have very high fan-out numbers is not a good design. The reason for this is that a very high fan-out is an indication that the module lacks cohesion. A module having a large fan-out (greater than 7) is likely to implement several different functions and not just a single cohesive function.

Fan-in: Fan-in indicates the number of modules that directly invoke a given module. High fan-in represents code reuse and is in general, desirable in a good design. In Figure 5.6(a), the fan-in of the module M1 is 0, that of M2 is 1, and that of M5 is 2.

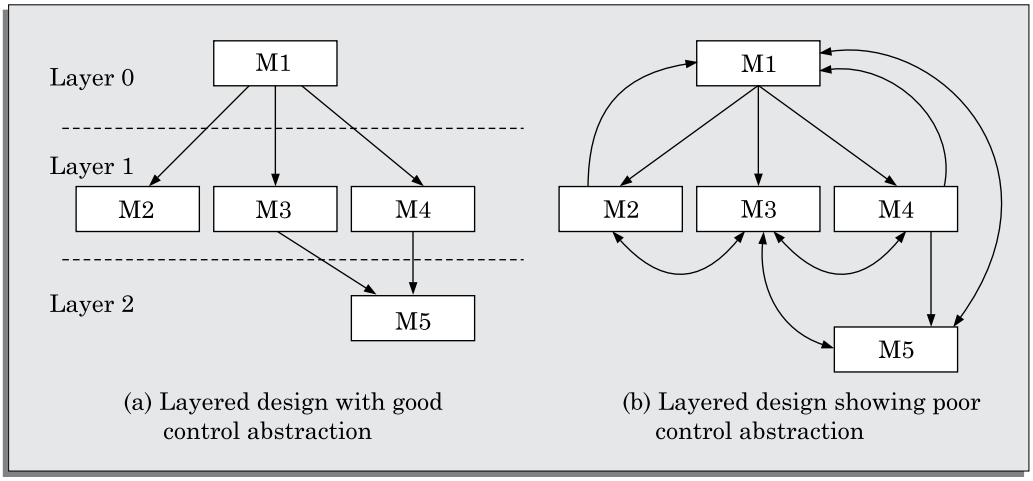


FIGURE 5.6 Examples of good and poor control abstraction.

5.5 APPROACHES TO SOFTWARE DESIGN

There are two fundamentally different approaches to software design that are in use today—function-oriented design, and object-oriented design. Though these two design approaches are radically different, they are complementary rather than competing techniques. The object-oriented approach is a relatively newer technology and is still evolving. For development of large programs, the object-oriented approach is becoming increasingly popular due to certain advantages that it offers. On the other hand, function-oriented designing is a mature technology and has a large following. Salient features of these two approaches are discussed in subsections 5.5.1 and 5.5.2 respectively.

5.5.1 Function-oriented Design

The following are the salient features of the function-oriented design approach:

Top-down decomposition: A system, to start with, is viewed as a black box that provides certain services (also known as high-level functions) to the users of the system.

In top-down decomposition, starting at a high-level view of the system, each high-level function is successively refined into more detailed functions.

For example, consider a function create-new-library member which essentially creates the record for a new member, assigns a unique membership number to him, and prints a bill towards his membership charge. This high-level function may be refined into the following subfunctions:

- `assign-membership-number`
- `create-member-record`
- `print-bill`

Each of these subfunctions may be split into more detailed subfunctions and so on.

Centralised system state: The system state can be defined as the values of certain data items that determine the response of the system to a user action or external event. For example, the set of books (i.e. whether borrowed by different users or available for issue) determines the state of a library automation system. Such data in procedural programs usually have global scope and are shared by many modules.

The system state is centralised and shared among different functions.

For example, in the library management system, several functions such as the following share data such as member-records for reference and updation:

- `create-new-member`
- `delete-member`
- `update-member-record`

A large number of function-oriented design approaches have been proposed in the past. A few of the well-established function-oriented design approaches are as following:

- Structured design by Constantine and Yourdon [1979]
- Jackson's structured design by Jackson [1975]
- Warnier-Orr methodology [1977, 1981]
- Step-wise refinement by Wirth [1971]
- Hatley and Pirbhai's Methodology [1987]

5.5.2 Object-oriented Design

In the *object-oriented design* (OOD) approach, a system is viewed as being made up of a collection of objects (i.e., entities). Each object is associated with a set of functions that are called its *methods*. Each object contains its own data and is responsible for managing it. The data internal to an object cannot be accessed directly by other objects and only through invocation of the methods of the object. The system state is decentralised since there is no globally shared data in the system and data is stored in each object. For example, in a library automation software, each library member may be a separate object with its own data and functions to operate on the stored data. The methods defined for one object cannot directly refer to or change the data of other objects.

The object-oriented design paradigm makes extensive use of the principles of abstraction and decomposition as explained below. Objects decompose a system into functionally independent modules. Objects can also be considered as instances of *abstract data types* (ADTs). The ADT concept did not originate from the object-oriented approach. In fact, ADT concept was extensively used in the ADA programming language introduced in the 1970s. ADT is an important concept that forms an important pillar of object-orientation. Let us

now discuss the important concepts behind an ADT. There are, in fact, three important concepts associated with an ADT—data abstraction, data structure, data type. We discuss these in the following subsection:

Data abstraction: The principle of data abstraction implies that how data is exactly stored is abstracted away. This means that any entity external to the object (that is, an instance of an ADT) would have no knowledge about how data is exactly stored, organised, and manipulated inside the object. The entities external to the object can access the data internal to an object only by calling certain well-defined methods supported by the object. Consider an ADT such as a `stack`. The data of a stack object may internally be stored in an array, a linearly linked list, or a bidirectional linked list. The external entities have no knowledge of this and can access data of a stack object only through the supported operations such as push and pop.

Data structure: A data structure is constructed from a collection of primitive data items. Just as a civil engineer builds a large civil engineering structure using primitive building materials such as bricks, iron rods, and cement; a programmer can construct a data structure as an organised collection of primitive data items such as integer, floating point numbers, characters, etc.

Data type: A type is a programming language terminology that refers to anything that can be instantiated. For example, `int`, `float`, `char`, etc., are the basic data types supported by C programming language. Thus, we can say that ADTs are user defined data types.

In object-orientation, classes are ADTs. But, what is the advantage of developing an application using ADTs? Let us examine the three main advantages of using ADTs in programs:

- The data of objects are *encapsulated* within the methods. The encapsulation principle is also known as *data hiding*. The encapsulation principle requires that data can be accessed and manipulated only through the methods supported by the object and not directly. This localises the errors. The reason for this is as follows. No program element is allowed to change a data, except through invocation of one of the methods. So, any error can easily be traced to the code segment changing the value. That is, the method that changes a data item, making it erroneous can be easily identified.
- An ADT-based design displays high cohesion and low coupling. Therefore, object-oriented designs are highly modular.
- Since the principle of abstraction is used, it makes the design solution easily understandable and helps to manage complexity.

Similar objects constitute a class. In other words, each object is a member of some class. Classes may inherit features from a super class. Conceptually, objects communicate by message passing. Objects have their own internal data. Thus an object may exist in different states depending the values of the internal data. In different states, an object may behave differently. We shall elaborate these concepts in Chapter 7 and subsequently we discuss an object-oriented design methodology in Chapter 8.

Object-oriented *versus* function-oriented design approaches

The following are some of the important differences between the function-oriented and object-oriented design:

- Unlike function-oriented design methods in OOD, the basic abstraction is not the services available to the users of the system such as `issue-book`, `display-book-details`, `find-issued-books`, etc., but real-world entities such as `member`, `book`, `book-register`, etc. For example in OOD, an employee pay-roll software is not developed by designing functions such as update-employee-record, get-employee-address, etc., but by designing objects such as employees, departments, etc.
- In OOD, state information exists in the form of data distributed among several objects of the system. In contrast, in a procedural design, the state information is available in a centralised shared data store. For example, while developing an employee pay-roll system, the employee data such as the names of the employees, their code numbers, basic salaries, etc., are usually implemented as global data in a traditional programming system; whereas in an object-oriented design, these data are distributed among different employee objects of the system. Objects communicate by message passing. Therefore, one object may discover the state information of another object by sending a message to it. Of course, somewhere or other the real-world functions must be implemented.
- Function-oriented techniques group functions together if, as a group, they constitute a higher level function. On the other hand, object-oriented techniques group functions together on the basis of the data they operate on.

To illustrate the differences between the object-oriented and the function-oriented design approaches, let us consider an example—that of an automated fire-alarm system for a large building.

Automated fire-alarm system—customer requirements

The owner of a large multi-storied building wants to have a computerised fire alarm system designed, developed, and installed in his building. Smoke detectors and fire alarms would be placed in each room of the building. The fire alarm system would monitor the status of these smoke detectors. Whenever a fire condition is reported by any of the smoke detectors, the fire alarm system should determine the location at which the fire has been sensed and then sound the alarms only in the neighbouring locations. The fire alarm system should also flash an alarm message on the computer console. Fire fighting personnel would man the console round the clock. After a fire condition has been successfully handled, the fire alarm system should support resetting the alarms by the fire fighting personnel.

Function-oriented approach: In this approach, the different high-level functions are first identified, and then the data structures are designed.

```
/* Global data (system state) accessible by various functions */
     BOOL   detector_status[MAX_ROOMS];
     int    detector_locs[MAX_ROOMS];
     BOOL   alarm-status[MAX_ROOMS]; /* alarm activated when status is set */
     int    alarm_locs[MAX_ROOMS]; /* room number where alarm is located */
     int    neighbour-alarms[MAX-ROOMS][10]; /* each detector has at most */
                                        /* 10 neighbouring alarm locations */
     int    sprinkler[MAX_ROOMS];
```

The functions which operate on the system state are:

```
interrogate_detectors();
get_detector_location();
```

```
determine_neighbour_alarm();
determine_neighbour_sprinkler();
ring_alarm();
activate_sprinkler();
reset_alarm();
reset_sprinkler();
report_fire_location();
```

Object-oriented approach: In the object-oriented approach, the different classes of objects are identified. Subsequently, the methods and data for each object are identified. Finally, an appropriate number of instances of each class is created.

```
class detector
attributes: status, location, neighbours
operations: create, sense-status, get-location, find-neighbours

class alarm
attributes: location, status
operations: create, ring-alarm, get_location, reset-alarm

class sprinkler
attributes: location, status
operations: create, activate-sprinkler, get_location, reset-sprinkler
```

We can now compare the function-oriented and the object-oriented approaches based on the two examples discussed above, and easily observe the following main differences:

- In a function-oriented program, the system state (data) is centralised and several functions access and modify this central data. In case of an object-oriented program, the state information (data) is distributed among various objects.
- In the object-oriented design, data is private in different objects and these are not available to the other objects for direct access and modification.
- The basic unit of designing an object-oriented program is objects, whereas it is functions and modules in procedural designing. Objects appear as nouns in the problem description; whereas functions appear as verbs.

At this point, we must emphasise that it is not necessary that an object-oriented design be implemented by using an object-oriented language only. However, an object-oriented language such as C++ and Java support the definition of all the basic mechanisms of class, inheritance, objects, methods, etc. and also support all key object-oriented concepts that we have just discussed. Thus, an object-oriented language facilitates the implementation of an OOD. However, an OOD can as well be implemented using a conventional procedural languages—though it may require more effort to implement an OOD using a procedural language as compared to the effort required for implementing the same design using an object-oriented language. In fact, the older C++ compilers were essentially pre-processors that translated C++ code into C code.

Even though object-oriented and function-oriented techniques are remarkably different approaches to software design, yet one does not replace the other; but they complement each other in some sense. For example, usually one applies the top-down function oriented techniques to design the internal methods of a class, once the classes are identified. In this case, though outwardly the system appears to have been developed in an object-oriented fashion, but inside each class there may be a small hierarchy of functions designed in a top-down manner.

SUMMARY

- A software design is typically carried out through two stages—high-level design, and detailed design. During high-level design, the important components (modules) of the system and their interactions are identified. During detailed design, the algorithms and data structures are identified.
- We discussed that there is no unique design solution to any problem and one needs to choose the best solution among a set of candidate solutions. To be able to achieve this, we identified the factors based on which a superior design can be distinguished from a inferior design.
- We discussed that understandability of a design is a major criterion determining the goodness of a design. We characterised the understandability of design in terms of satisfactory usage of decomposition and abstraction principles. Later, we characterised these in terms of cohesion, coupling, layering, control abstraction, fan-in, fan-out, etc.
- We identified two fundamentally different approaches to software design—function-oriented design and object-oriented design. We discussed the essential philosophy governing these two approaches and argued that these two approaches to software design are not really competing approaches but complementary approaches.

EXERCISES

MULTIPLE CHOICE QUESTIONS

Choose the correct option:

1. The extent of data interchange between two modules is called:
(a) Coupling
(b) Cohesion
(c) Structure
(d) Union

2. Which of the following types of cohesion can be considered as the strongest cohesion?
(a) Logical
(b) Coincidental
(c) Temporal
(d) Functional

3. The modules in a good software design should have which of the following characteristics:
(a) High cohesion, low coupling
(b) Low cohesion, high coupling
(c) Low cohesion, low coupling
(d) High cohesion, high coupling

TRUE OR FALSE

State whether the following statements are **TRUE** or **FALSE**. Give reasons for your answer.

1. The essence of any good function-oriented design technique is to map the functions performing similar activities into a module.
2. Traditional procedural design is carried out top-down whereas object-oriented design is normally carried out bottom-up.
3. Common coupling is the worst type of coupling between two modules.
4. Temporal cohesion is the worst type of cohesion that a module can have.
5. The extent to which two modules depend on each other determines the cohesion of the two modules.

REVIEW QUESTIONS

1. Do you agree with the following assertion? A design solution that is difficult to understand would lead to increased development and maintenance cost. Give reasonings for your answer.
2. What do you mean by the terms cohesion and coupling in the context of software design? How are these concepts useful in arriving at a good design of a system?
3. What do you mean by a modular design? How can you determine whether a given design is modular or not?
4. Enumerate the different types of cohesion that a module in a design might exhibit. Give examples of each.
5. Enumerate the different types of coupling that might exist between two modules. Give examples of each.
6. Is it true that whenever you increase the cohesion of your design, coupling in the design would automatically decrease? Justify your answer by using suitable examples.
7. What according to you are the characteristics of a good software design?
8. What do you understand by the term functional independence in the context of software design? What are the advantages of functional independence? How can functional independence in a software design be achieved?
9. Explain how the principles of abstraction and decomposition are used to arrive at a good design.
10. What do you understand by information hiding in the context of software design? Explain why a design approach based on the information hiding principle is likely to lead to a reusable and maintainable design. Illustrate your answer with a suitable example.
11. In the context of software development, distinguish between analysis and design with respect to intention, methodology, and the documentation technique used.
12. Compare relative advantages of the object-oriented and function-oriented approaches to software design.

13. Name a few well-established function-oriented software design techniques.

14. Explain the important causes of and remedies for high coupling between two software modules.

15. What problems are likely to arise if two modules have high coupling?

16. What problems are likely to occur if a module has low cohesion?

17. Distinguish between high-level and detailed designs. What documents should be produced on completion of high-level and detailed designs respectively?

18. What is meant by the term cohesion in the context of software design? Is it true that in a good design, the modules should have low cohesion? Why?

19. What is meant by the term coupling in the context of software design? Is it true that in a good design, the modules should have low coupling? Why?

20. What do you mean by modular design? What are the different factors that affect the modularity of a design? How can you assess the modularity of a design? What are the advantages of a modular design?

21. How would you improve a software design that displays very low cohesion and high coupling?

22. Explain how the overall cohesion and coupling of a design would be impacted if all modules of the design are merged into a single module.

23. Explain what do you understand by the terms decomposition and abstraction in the context of software design. How are these two principles used in arriving good procedural designs?

24. What is an ADT? What advantages accrue when a software design technique is based on ADTs? Explain why the object paradigm is said to be based on ADTs.

25. By using suitable examples explain the following terms associated with an abstract data type (ADT)—data abstraction, data structure, data type.

26. What do you understand by the term top-down decomposition in the context of function-oriented design? Explain your answer using a suitable example.

27. What do you understand by a layered software design? What are the advantages of a layered design? Explain your answer by using suitable examples.

28. What is the principal difference between the software design methodologies based on functional abstraction and those based on data abstraction? Name at least one popular design technique based on each of these two software design paradigms.

29. What are the main advantages of using an object-oriented approach to software design over a function-oriented approach?

30. Point out three important differences between the function oriented and the object-oriented approaches to software design. Corroborate your answer through suitable examples.

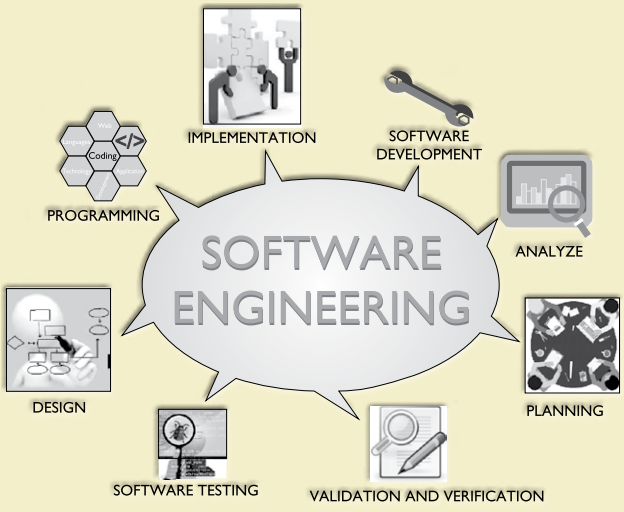
31. Identify the criteria that you would use to decide which one of two alternate function-oriented design solutions to a problem is superior.

32. Explain the main differences between architectural design, high-level-design, and detailed design of a software system.

CHAPTER

6

FUNCTION-ORIENTED SOFTWARE DESIGN



CHAPTER OUTLINE

- Overview of SA/SD Methodology
- Structured Analysis
- Developing the DFD Model of a System
- Structured Design
- Detailed Design
- Design Review

The belief that complex systems require armies of designers and programmers is wrong. A system that is not understood in its entirety, or at least to a significant degree of detail by a single individual, should probably not be built.

—Niklaus Wirth

LEARNING OBJECTIVES

- Basic concepts in structured analysis and structured design
- Development of the DFD model of a system
- Transformation of DFD model into a structure chart model

Function-oriented design techniques were proposed nearly four decades ago. These techniques are at the present time still very popular and are currently being used in many software development projects. 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 provided by a software (e.g., `issue book, serach 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 detailed functions.

The term *top-down decomposition* is often used to denote the successive decomposition of a set of high-level functions into more detailed functions.

After top-down decomposition has been carried out, the different identified functions are mapped to modules and a module structure is created. This module structure would possess all the characteristics of a good design identified in the last chapter.

In this text, we shall not focus on any specific design methodology. Instead, we shall discuss a methodology that has the essential features of several important function-oriented design methodologies. Such an approach shall enable us to easily assimilate any specific design methodology in the future whenever the need arises. Learning a specific methodology may become necessary for you later, since different software development houses follow different methodologies. After all, the different procedural design techniques can be considered as sister techniques that have only minor differences with respect to the methodology and notations. We shall call the design technique discussed in this text as *structured analysis/structured design* (SA/SD) methodology. This technique draws heavily from the design methodologies proposed by the following authors:

- DeMarco and Yourdon [1978]
- Constantine and Yourdon [1979]
- Gane and Sarson [1979]
- Hatley and Pirbhai [1987]

The SA/SD technique can be used to perform the high-level design of a software. The details of SA/SD technique are discussed further.

6.1 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 Figure 6.1. Observe the following from the figure:

- 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 structure chart.

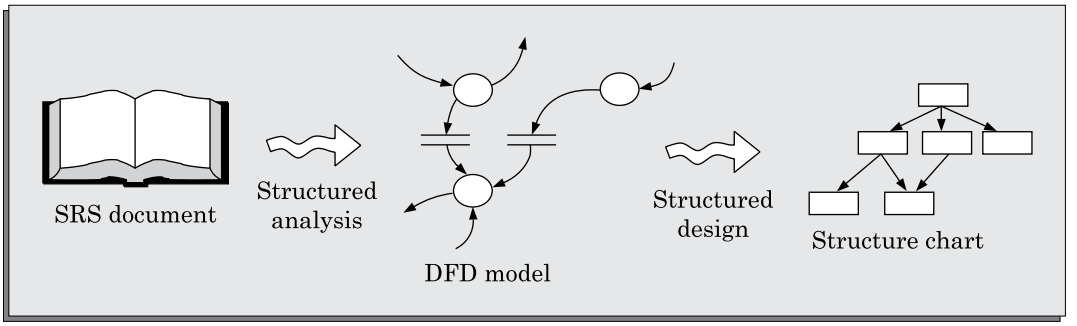


FIGURE 6.1 Structured analysis and structured design methodology.

As shown in Figure 6.1, 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. That is, each function that the system needs to perform is analysed and hierarchically decomposed into more detailed functions. On the other hand, 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 high-level design stage is normally followed by a detailed design stage. 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 programming language.

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 his requirements.

In the following section, we first discuss how to carry out structured analysis to construct the DFD model. Subsequently, we discuss how the DFD model can be transformed into structured design.

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 programming language.

6.2 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 represented graphically. Significant contributions to the development of the structured analysis techniques have been made by Gane and Sarson [1979], and DeMarco and Yourdon [1978]. 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).

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 standard problems.

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 be executed. In fact, it completely ignores aspects such as control flow, the specific algorithms used by the functions, etc. In the DFD terminology, each function is called a *process* or a *bubble*. It is useful to consider each function as a processing station (or process) that consumes some input data and produces some output data.

DFD is an elegant modelling technique that can be used not only to represent the results of structured analysis of a software problem, but also useful for several other applications such as showing the flow of documents or items in an organisation. Recall that in Chapter 1 we had given an example (see Figure 1.10) to illustrate how a DFD can be used to represent the processing activities and flow of material in an automated car assembling plant. We now elaborate how a DFD model can be constructed.

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.

6.2.1 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 the system. 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 and use. 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 these functions.

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. We had pointed out while discussing the principle of abstraction in Section 1.3.2 that any hierarchical representation is an effective means to tackle complexity. 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. We now elaborate the different concepts associated with building a DFD model of a system.

Primitive symbols used for constructing DFDs

There are essentially five different types of symbols used for constructing DFDs. These primitive symbols are depicted in Figure 6.2. The meaning of these symbols are explained as follows:

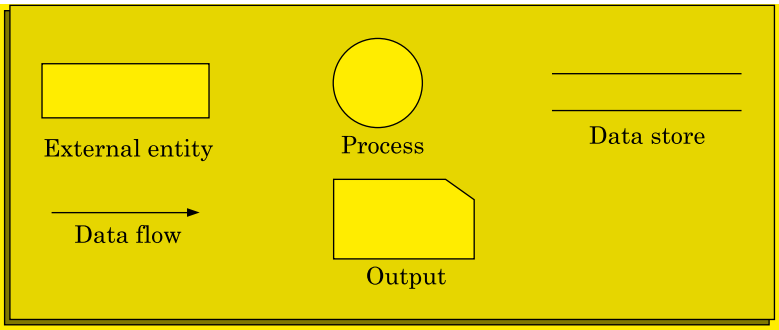


FIGURE 6.2 Symbols used for designing DFDs.

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 (see Figure 6.3).

External entity symbol: An external entity such as a librarian, a library member, etc. is 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. 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 being modelled.

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 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 of `validate-number`.

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. 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 to 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 Figure 6.3(b).

Output symbol: The output symbol is as shown in Figure 6.2. The output symbol is used when a hard copy is produced.

The notations that we are following in this text are closer to the Yourdon's notations than to the other notations. You may sometimes find notations in other books that are slightly different than those discussed here. For example, the data store may look like a box with one end open. That is because, they may be following notations such as those of Gane and Sarson [1979].

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 with DFDs:

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 Figure 6.3(a). 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 its data.

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 of them.

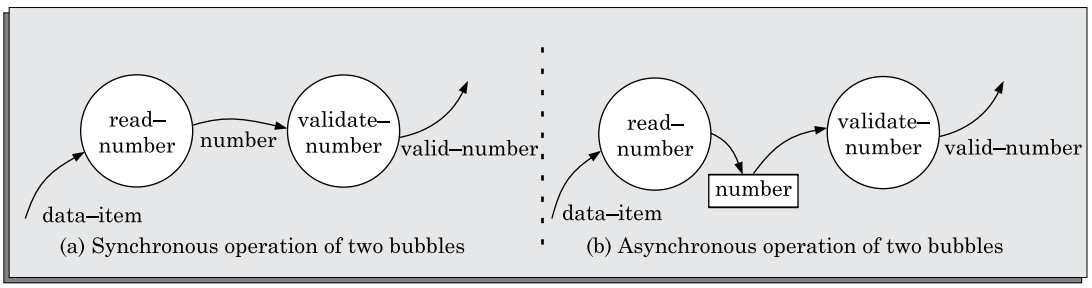


FIGURE 6.3 Synchronous and asynchronous data flow.

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. The data items listed include all data flows and the contents of all data stores appearing on all the DFDs in a DFD model. Please remember that

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.

the DFD model of a system typically consists of several DFDs, viz., level 0 DFD, level 1 DFD, level 2 DFDs, etc., as shown in Figure 6.4 discussed in new subsection. However, a single data dictionary should capture all the data appearing in all the DFDs constituting the DFD model of a system.

For example, a data dictionary entry may represent that the data *grossPay* consists of the components *regularPay* and *overtimePay*.

$$grossPay = regularPay + overtimePay$$

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 data items are discussed subsequently.

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. A consistent vocabulary for data items is very important, since in large projects different developers of the project have a tendency to use different terms to refer to the same data, which unnecessarily causes confusion.
- 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*. Such impact analysis is especially useful when one wants to check the impact of changing an input value type, or a bug in some functionality, etc.

For large systems, the data dictionary can become extremely complex and voluminous. Even 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 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. As a result, the designers do not have to spend almost any effort in creating the data dictionary. These CASE tools also support some query language facility to query about the definition and usage of data items. 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 definition operators.

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

`[,,]` : 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+b` occurs.

`{}` : represents iterative data definition, e.g. `{name}5` represents five `name` data. `{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` and `c`.

`/**/` : Anything appearing within `/*` and `*/` is considered as comment.

6.3 DEVELOPING THE DFD MODEL OF A SYSTEM

A DFD model of a system graphically represents how each input data is transformed to its corresponding output data through a hierarchy of DFDs.

The DFD model of a system is constructed by using a hierarchy of DFDs (see Figure 6.4). 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.

> The DFD model of a problem consists of many DFDs and a single data dictionary.

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 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. All the data names appearing in all DFDs are populated in the data dictionary and the data dictionary contains the definitions of all the data items.

6.3.1 Context Diagram

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 (usually a noun). This is the only bubble in a DFD model, where a noun is used for naming the bubble. The bubbles at all other levels are annotated with verbs according to the main function performed by the bubble. 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 'Supermarket software'.

> 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 receive from the system.

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 the system and the specific data items that they would be supplying the system and

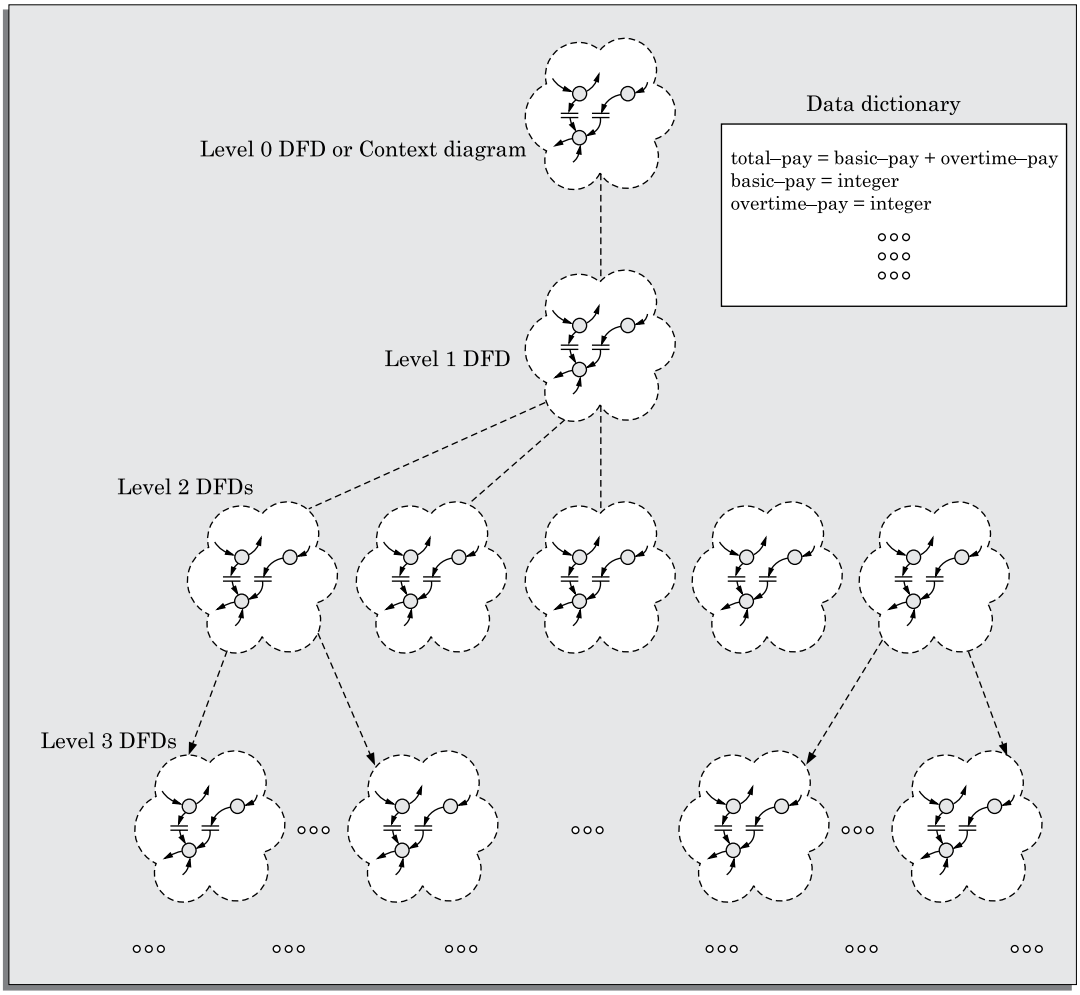


FIGURE 6.4 DFD model of a system consists of a hierarchy of DFDs and a single data dictionary.

the data items they would be receiving from the system. The various external entities with which the system interacts and the data flow occurring between the system and the external entities are represented. The data input to the system and the data output from the system are represented as incoming and outgoing arrows. These data flow arrows should be annotated with the corresponding data names.

To develop the context diagram of the system, we have to analyse the SRS document to identify the different types of users who would be using the system and the kinds of data they would be inputting to the system and the data they would be receiving from the system. Here, the term users of the system also includes any external systems which supply data to or receive data from the system.

6.3.2 Level 1 DFD

The level 1 DFD usually contains three to seven bubbles. That is, 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.

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 in the level 1 DFD. 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 subfunctions so that we have roughly about five to seven bubbles represented on the diagram. We illustrate construction of level 1 DFDs in Examples 6.1 to 6.4.

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* or *exploding* a bubble. Each bubble at any level of DFD is usually decomposed to anything three to seven bubbles. A few bubbles at any level make that level superfluous. For example, if a bubble is decomposed to just one bubble or two bubbles, then this decomposition becomes trivial and redundant. On the other hand, too many bubbles (i.e. more than seven bubbles) at any level of a DFD makes the DFD model hard to understand. 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.

We can now describe how to go about developing the DFD model of a system more systematically.

1. **Construction of context diagram:** Examine the SRS document to determine:
   - Different high-level functions that the system needs to perform.
   - Data input to every high-level function.
   - Data output from every high-level function.
   - Interactions (data flow) among the identified high-level functions.

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 DFD 0.

2. **Construction of level 1 diagram:** Examine the high-level functions described in the SRS document. 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 a bubble. If there are more than seven bubbles, then some of them have to be combined. If there are less than three bubbles, then some of these have to be split.
3. **Construction of lower-level diagrams:** Decompose each high-level function into its constituent subfunctions through the following set of activities:

- Identify the different subfunctions of the high-level function.
- Identify the data input to each of these subfunctions.
- Identify the data output from each of these subfunctions.
- Identify the interactions (data flow) among these subfunctions.

Represent these aspects in a diagrammatic form using a DFD.
Recursively repeat Step 3 for each subfunction until a subfunction can be represented by using a simple algorithm.

Numbering of bubbles

It is necessary to number the different bubbles occurring in the DFD. 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 level DFD. 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 its successors.

Balancing DFDs

The DFD model of a system usually consists of many DFDs that are organised in a hierarchy. In this context, a DFD is required to be balanced with respect to the corresponding bubble of the parent DFD.

We illustrate the concept of balancing a DFD in Figure 6.5. 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 dotted circle). 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. Please note that dangling arrows (d1, d2, d3) represent the data flows into or out of a diagram.

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* a DFD.

How far to decompose?

A bubble should not be decomposed any further once a bubble is found to represent a simple set of instructions. For simple problems, decomposition up to level 1 should suffice. However, large industry standard problems may need decomposition up to level 3 or level 4. Rarely, if ever, decomposition beyond level 4 is needed.

Commonly made errors while constructing a DFD model

Although DFDs are simple to understand and draw, students and practitioners alike encounter similar types of problems while modelling software problems using DFDs. While learning from experience is a powerful thing, it is an expensive pedagogical technique in the business world. It is therefore useful to understand the different types of mistakes that beginners usually make while constructing the DFD model of systems, so that you can consciously try to avoid them. The errors are as follows:

- Many beginners commit the mistake of drawing more than one bubble in the context diagram. Context diagram should depict the system as a single bubble.

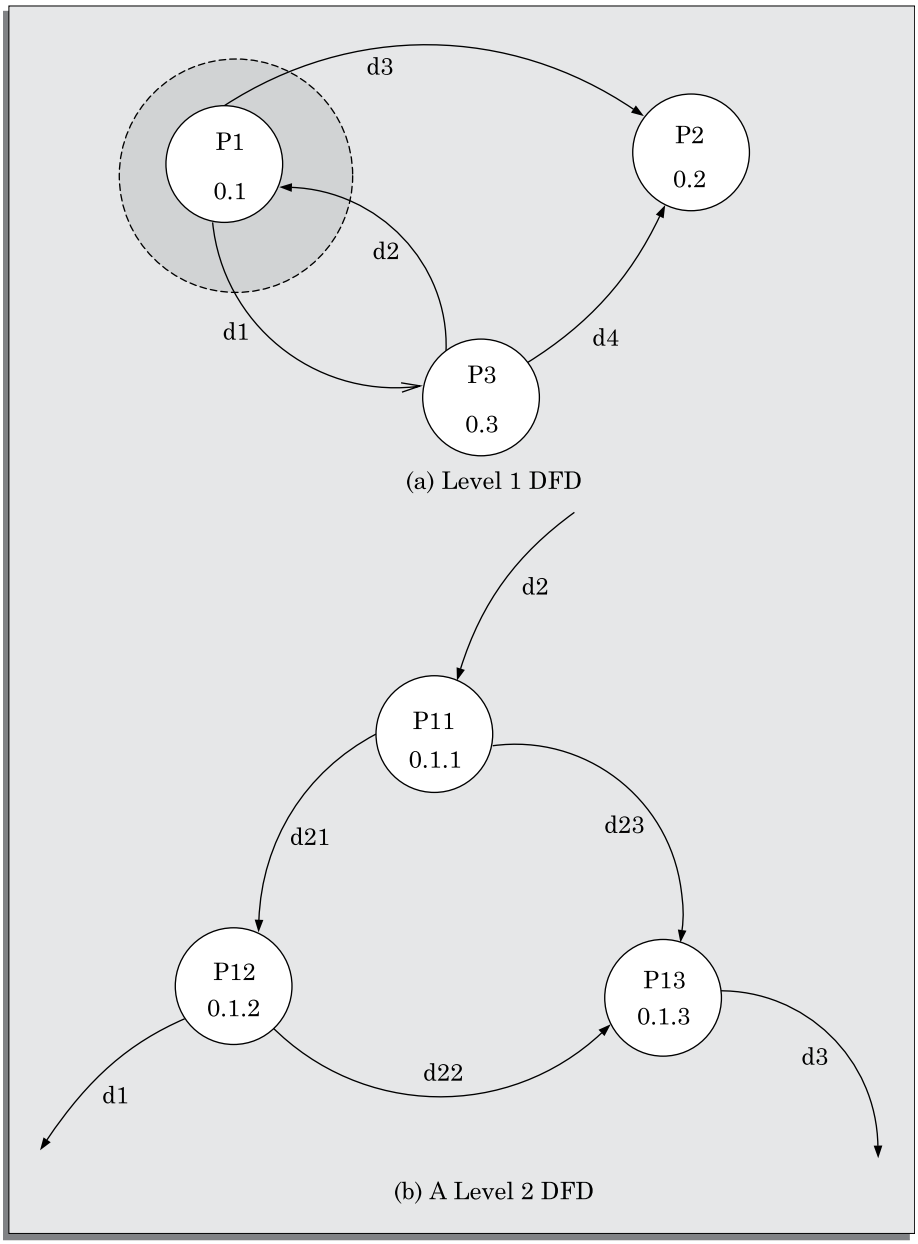


FIGURE 6.5 An example showing balanced decomposition.

- Many beginners create DFD models in which external entities appearing at all levels of DFDs. All external entities interacting with the system should be represented only in the context diagram. The external entities should not appear in the DFDs at any other level.

- It is a common oversight to have either too few or too many bubbles in a DFD. Only three to seven bubbles per diagram should be allowed. This also means that each bubble in a DFD should be decomposed three to seven bubbles in the next level.
- Many beginners leave the DFDs at the different levels of a DFD model unbalanced.
- A common mistake committed by many beginners while developing a DFD model is attempting to represent control information in a DFD.

It is important to realise that a DFD represents only data flow, and it does not represent any control information.

The following are some illustrative mistakes of trying to represent control aspects such as:

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.

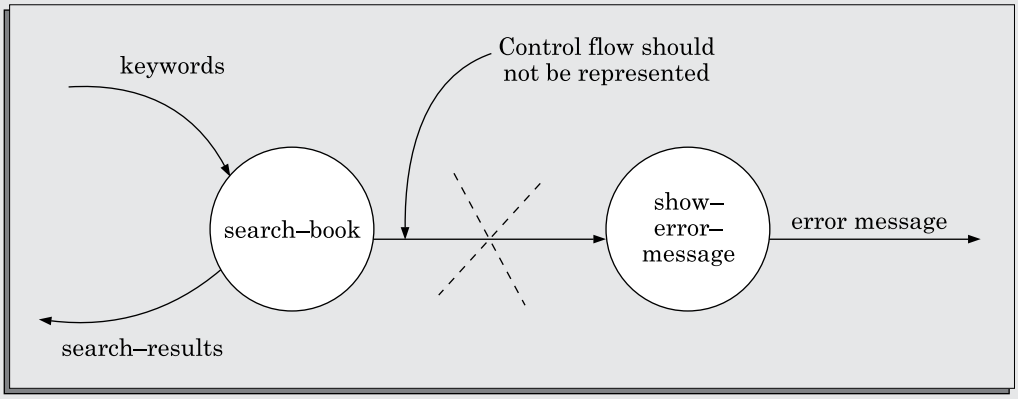


FIGURE 6.6 It is incorrect to show control information on a DFD.

ILLUSTRATION 2 Another type of error occurs when one tries to represent when or in what order difierent functions (processes) are invoked. A DFD similarly should not represent the conditions under which different functions are invoked.

ILLUSTRATION 3 If a bubble A invokes either the bubble B or the bubble C depending upon some conditions, we need only to represent the data that flows between bubbles A and B or bubbles A and C and not the conditions depending on which the two modules are invoked.

- A data flow arrow should not connect two data stores or even a data store with an external entity. Thus, data cannot flow from a data store to another data store or to an external entity without any intervening processing. As a result, a data store should be connected only to bubbles through data flow arrows.

- All the functionalities of the system must be captured by the DFD model. No function of the system specified in the SRS document of the system should be overlooked.
- Only those functions of the system specified in the SRS document should be represented. That is, the designer should not assume functionality of the system not specified by the SRS document and then try to represent them in the DFD.
- Incomplete data dictionary and data dictionary showing incorrect composition of data items are other frequently committed mistakes.
- The data and function names must be intuitive. Some students and even practicing developers use meaningless symbolic data names such as a,b,c, etc. Such names hinder understanding the DFD model.
- Novices usually clutter their DFDs with too many data flow arrow. It becomes difficult to understand a DFD if any bubble is associated with more than seven data flows. When there are too many data flowing in or out of a DFD, it is better to combine these data items into a high-level data item. Figure 6.7 shows an example concerning how a DFD can be simplified by combining several data flows into a single high-level data flow.

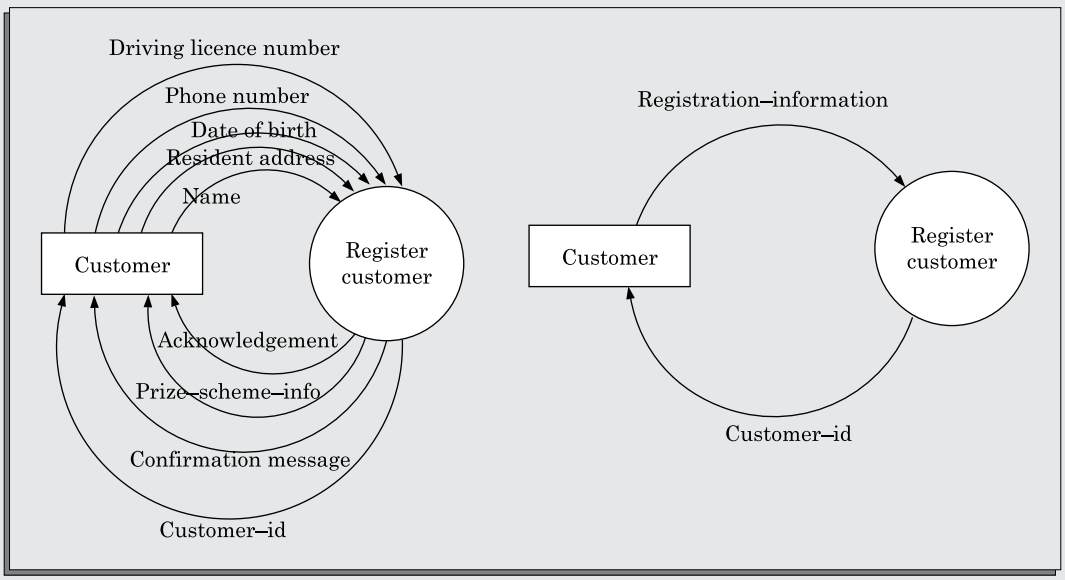


FIGURE 6.7 Illustration of how to avoid data cluttering.

We now illustrate the structured analysis technique through a few examples.

EXAMPLE 6.1 (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 display it.

In this example, the context diagram is simple to draw. The system accepts three integers from the user and returns the result to him. This has been shown in Figure 6.8(a). To draw the level 1 DFD, from a cursory analysis of the problem description, we can see that there are four basic functions that the system needs to perform—accept the input numbers from the user, validate the numbers, calculate the root mean square of the input numbers and, then display the result. After representing these four functions in Figure 6.8(b), we observe that the calculation of root mean square essentially consists of the functions—calculate the squares of the input numbers, calculate the mean, and finally calculate the root. This decomposition is shown in the level 2 DFD in Figure 6.8(c).

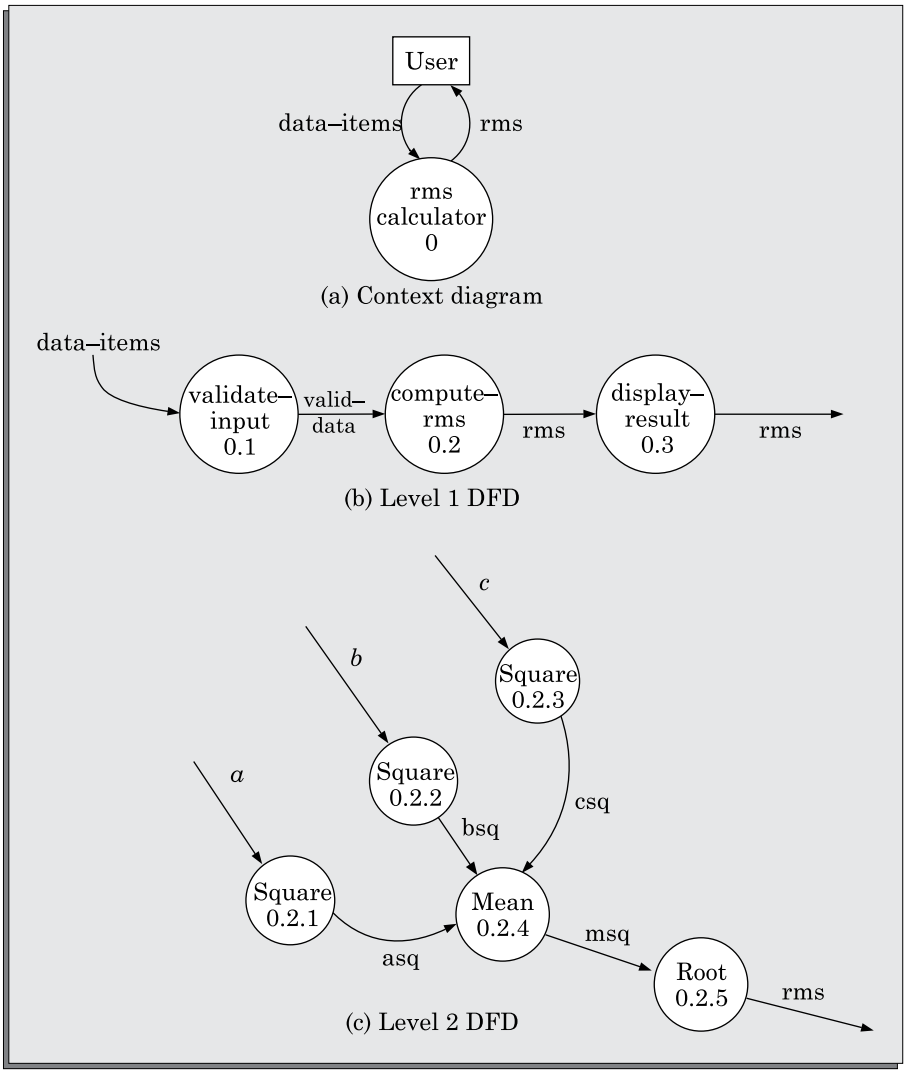


FIGURE 6.8 Context diagram, level 1, and level 2 DFDs for Example 6.1.

Data dictionary for the DFD model of Example 6.1

```
data-items: {integer}3
rms: float
valid-data:data-items
a: integer
b: integer
c: integer
asq: integer
bsq: integer
csq: integer
msq: integer
```

Example 6.1 is an almost trivial example and is only meant to illustrate the basic methodology. Now, let us perform the structured analysis for a more complex problem.

EXAMPLE 6.2 **(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 × 3 square. A move consists of marking a previously unmarked square. The player who is first to place three consecutive marks along a straight line (i.e., along a row, column, or diagonal) on the square wins. As soon as either of the human player or the computer wins, a message congratulating the winner should be displayed. 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 is drawn. The computer always tries to win a game.

The context diagram and the level 1 DFD are shown in Figure 6.9.

Data dictionary for the DFD model of Example 6.2

move: integer /* number between 1 to 9 */
display: game+result
game: board
board: {integer}9
result: [“computer won”, “human won”, “drawn”]

EXAMPLE 6.3 **(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 regular customers. In this scheme, a customer would have first register by supplying his/her residence address, telephone number, and the driving license number. 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 checkout staff when he makes any purchase. In this case, the value of his purchase is credited against his CN. At the end of each year, the supermarket intends to award surprise gifts to 10 customers who make the highest total purchase over the year. Also, it intends to award a 22 caret gold coin to every customer whose purchase exceeded ₹10,000. The entries against the CN are reset on the last day of every year after the prize winners’ lists are generated.

The context diagram for the supermarket prize scheme problem of Example 6.3 is shown in Figure 6.10. The level 1 DFD in Figure 6.11. The level 2 DFD in Figure 6.12.

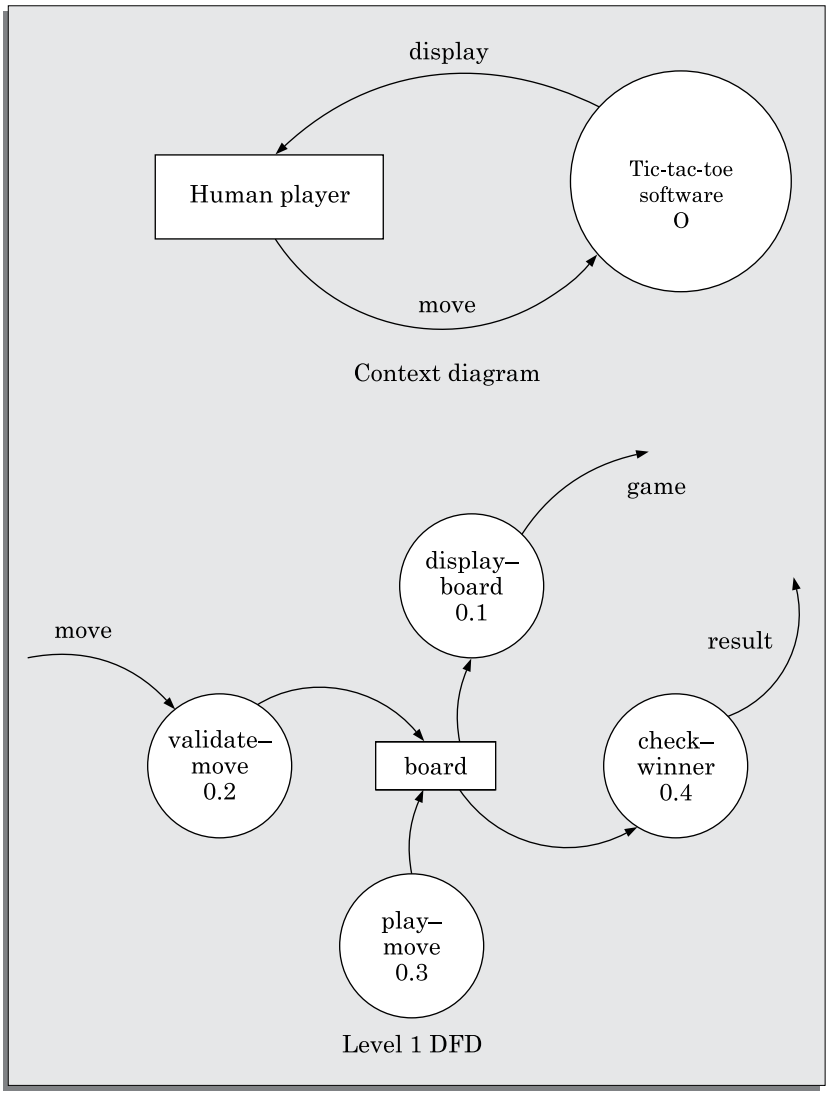


FIGURE 6.9 Context diagram and level 1 DFDs for Example 6.2.

Data dictionary for the DFD model of Example 6.3

address: name+house#+street#+city+pin
sales-details: {item+amount}* + CN
CN: integer
customer-data: {address+CN}*
sales-info: {sales-details}*
winner-list: surprise-gift-winner-list + gold-coin-winner-list

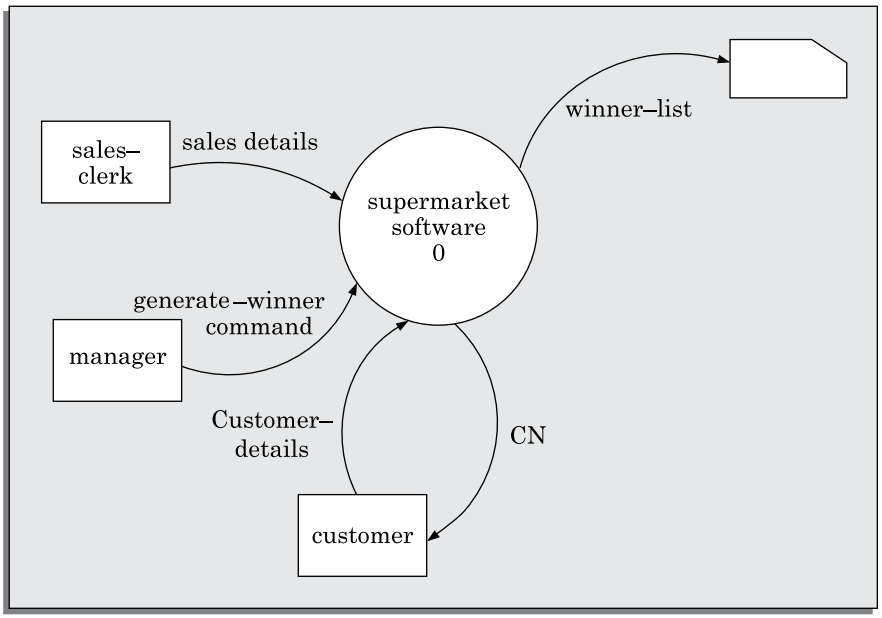


FIGURE 6.10 Context diagram for Example 6.3.

surprise-gift-winner-list: {address+CN}*
gold-coin-winner-list: {address+CN}*
gen-winner-command: command
total-sales: {CN+integer}*

Observations: The following observations can be made from the Example 6.3.

1. The fact that the customer is issued a manually prepared customer identity card or that the customer hands over the identity card each time he makes a purchase has not been shown in the DFD. This is because these are item transfers occurring outside the computer.
2. The data `generate-winner-list` in a way represents control information (that is, command to the software) and no real data. We have included it in the DFD because it simplifies the structured design process as we shall realize after we practise solving a few problems. We could have also as well done without the `generate-winner-list` data, but this could have a bit complicated the design.
3. Observe in Figure 6.11 that we have two separate stores for the customer data and sales data. Should we have combined them into a single data store? The answer is—No, we should not. If we had combined them into a single data store, the structured design that would be carried out based on this model would become complicated. Customer data and sales data have very different characteristics. For example, customer data once created, does not change. On the other hand, the sales data changes frequently and also the sales data is reset at the end of a year, whereas the customer data is not.

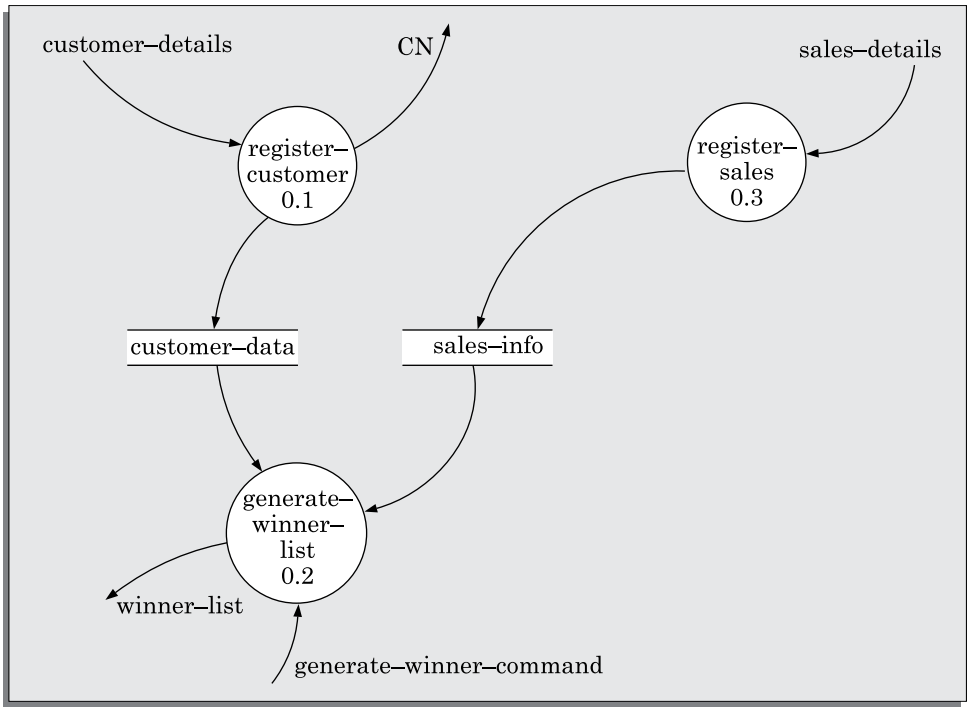


FIGURE 6.11 Level 1 diagram for Example 6.3.

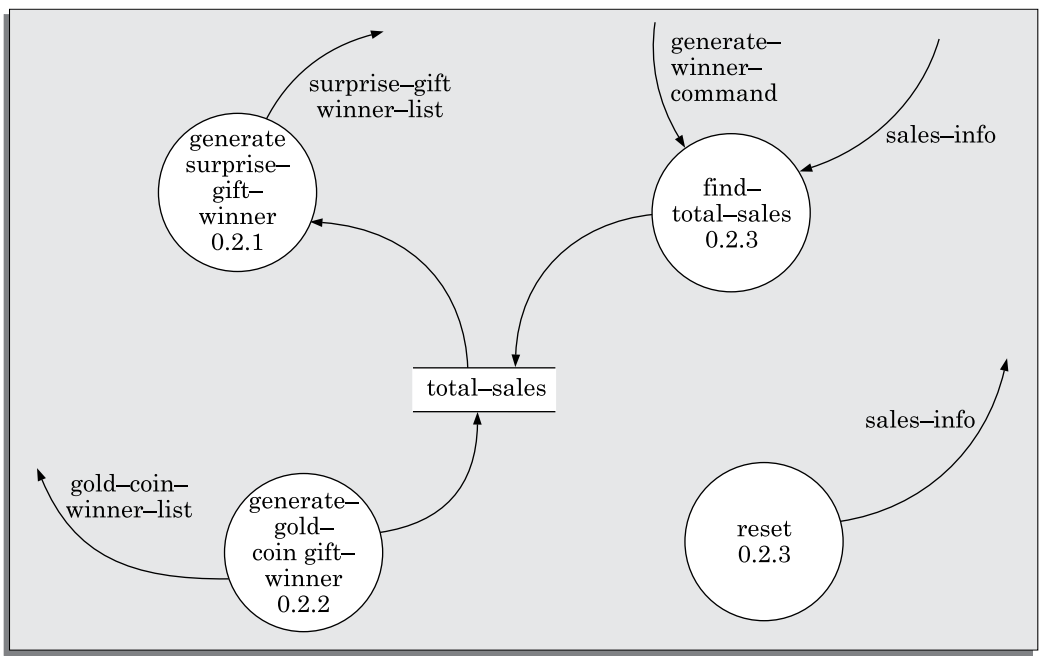


FIGURE 6.12 Level 2 diagram for Example 6.3.

EXAMPLE 6.4 **[Trading-house Automation System (TAS)]** A trading house wants us to develop a computerized system that would automate various book-keeping activities associated with its business. The following are the salient features of the system to be developed:

- The trading house has a set of regular customers. The customers place orders with it for various kinds of commodities. The trading house maintains the names and addresses of its regular customers. Each of these regular customers should be assigned a unique *customer identification number* (CIN) by the computer. The customers quote their CIN on every order they place.
- Once order is placed, as per current practice, the accounts department of the trading house first checks the credit-worthiness of the customer. The credit-worthiness of the customer is determined by analysing the history of his payments to different bills sent to him in the past. After automation, this task has be done by the computer.
- If a customer is not credit-worthy, his orders are not processed any further and an appropriate order rejection message is generated for the customer.
- If a customer is credit-worthy, the items that he has ordered are checked against the list of items that the trading house deals with. 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 is generated.
- The items in the customer's order that the trading house deals with are checked for availability in the inventory. If the items are available in the inventory in desired quantity, then:
  - A bill is with the forwarding address of the customer is printed.
  - A material issue slip is printed. The customer can produce this material issue slip at the store house and take delivery of the items.
  - Inventory data is adjusted to reflect the sale to the customer.
- 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 these vendors.
- 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 Figure 6.14.

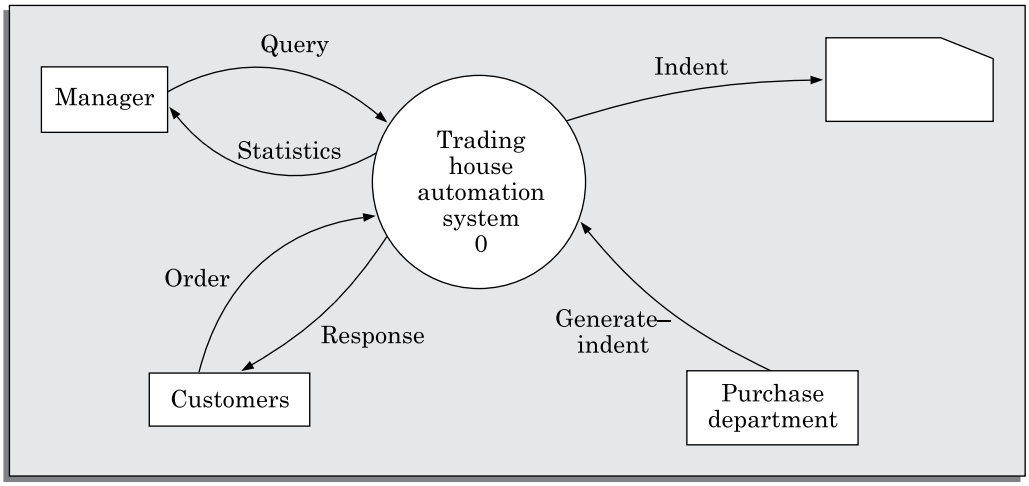


FIGURE 6.13 Context diagram for Example 6.4.

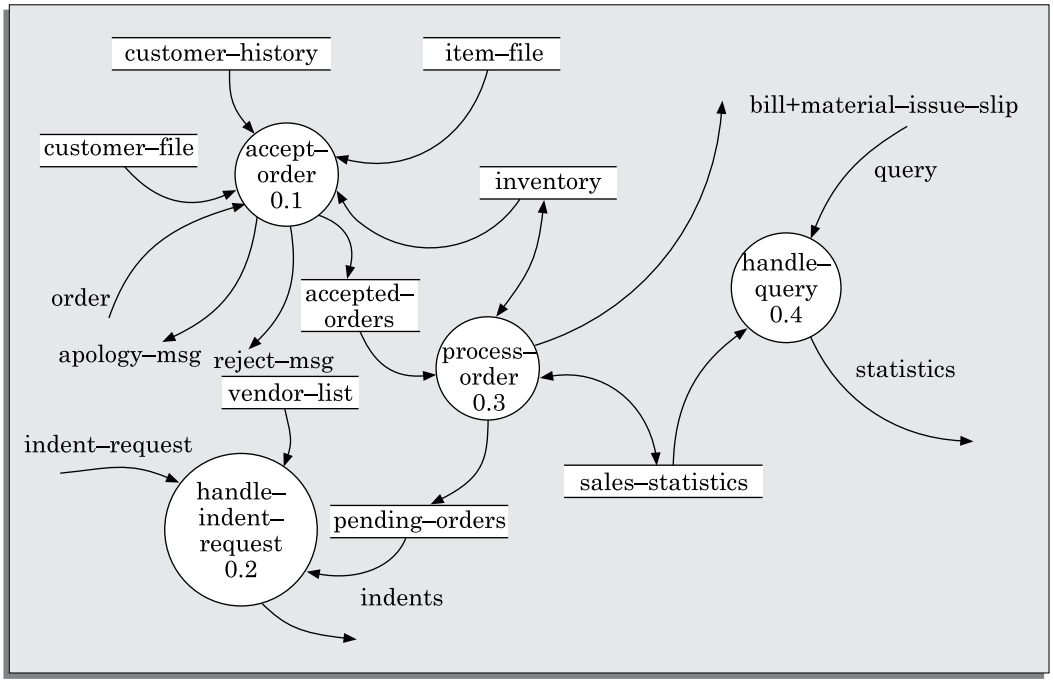


FIGURE 6.14 Level 1 DFD for Example 6.4.

Data dictionary for the DFD model of Example 6.4

response: [bill + material-issue-slip, reject-msg,apology-msg]
query: period /* query from manager regarding sales statistics*/
period: [date+date,month,year,day]

date: year + month + day
year: integer
month: integer
day: integer
customer-id: integer
order: customer-id + {items + quantity}* + order#
accepted-order: order /* ordered items available in inventory */
reject-msg: order + message /* rejection message */
pending-orders: customer-id + order# + {items+quantity}*
customer-address: name+house#+street#+city+pin
name: string
house#: string
street#: string
city: string
pin: integer
customer-id: integer
customer-file: {customer-address}* + customer-id
bill: {item + quantity + price}* + total-amount + customer-address + order#
material-issue-slip: message + item + quantity + customer-address
message: string
statistics: {item + quantity + price }*
sales-statistics: {statistics}* + date
quantity: integer
order#: integer /* unique order number generated by the program */
price: integer
total-amount: integer
generate-indent: command
indent: {item+quantity}* + vendor-address
indents: {indent}*
vendor-address: customer-address
vendor-list: {vendor-address}*
item-file: {item}*
item: string
indent-request: command

Observations: The following observations can be made from Example 6.4.

1. In a DFD, if two data stores deal with different types of data, e.g. one type of data is invariant with time whereas another varies with time, (e.g. vendor address, and inventory data) it is a good idea to represent them as separate data stores.

   The inventory data changes each time supply arrives and the inventory is updated or an item is sold, whereas the vendor data remains unchanged.

   If two types of data always get updated at the same time, they should be stored in a single data store. Otherwise, separate data stores should be used for them.

2. If we are developing the DFD model of a process which is already being manually carried out, then the names of the registers being maintained in the manual process would appear as data stores in the DFD model. For example, if TAS is currently

being manually carried out, then normally there would registers corresponding to accepted orders, pending orders, vendor list, etc.

3. We can observe that DFDs enable a software developer to develop the data domain and functional domain model of the system at the same time. As the DFD is refined into greater levels of detail, the analyst performs an implicit functional decomposition. At the same time, the DFD refinement automatically results in refinement of corresponding data items.
4. The data that are maintained in physical registers in manual processing, become data stores in the DFD representation. Therefore, to determine which data should be represented as a data store, it is useful to try to imagine whether a set of data items would be maintained in a register in a manual system.

EXAMPLE 6.5 **(Personal Library Software)** Structured analysis for the personal library software is shown in Figure 6.15, Figure 6.16 and Figure 6.17.

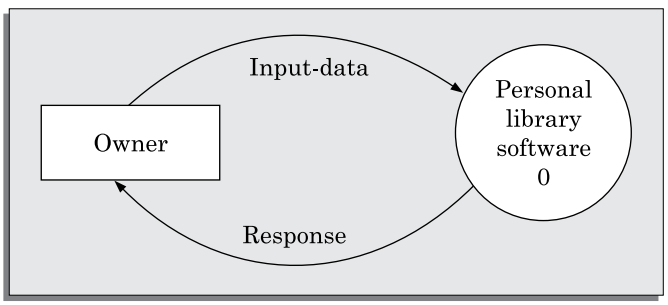


FIGURE 6.15 Context diagram for Example 6.5.

The level 1 DFD is shown in Figure 6.16.

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

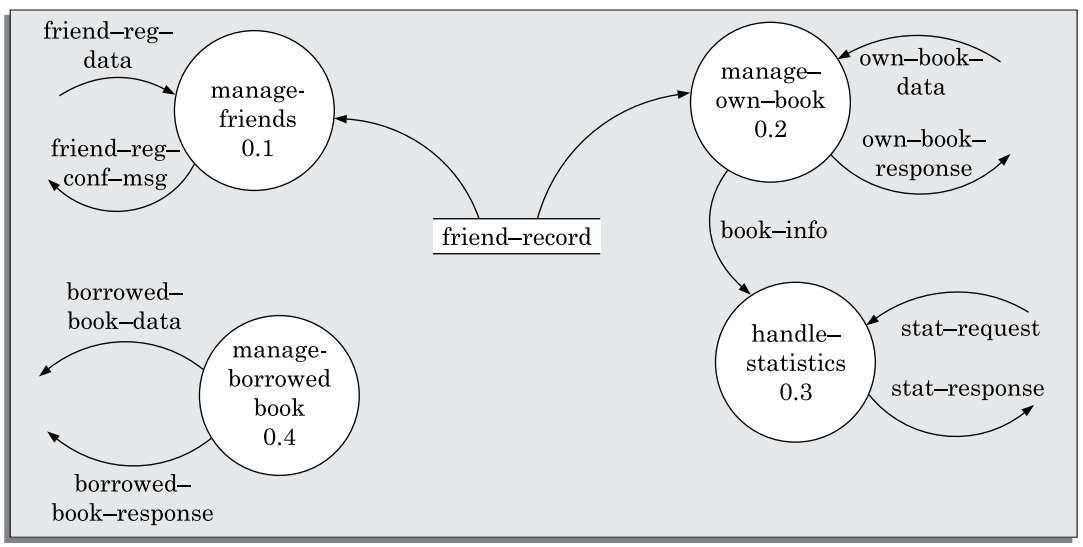


FIGURE 6.16 Level 1 DFD for Example 6.5.

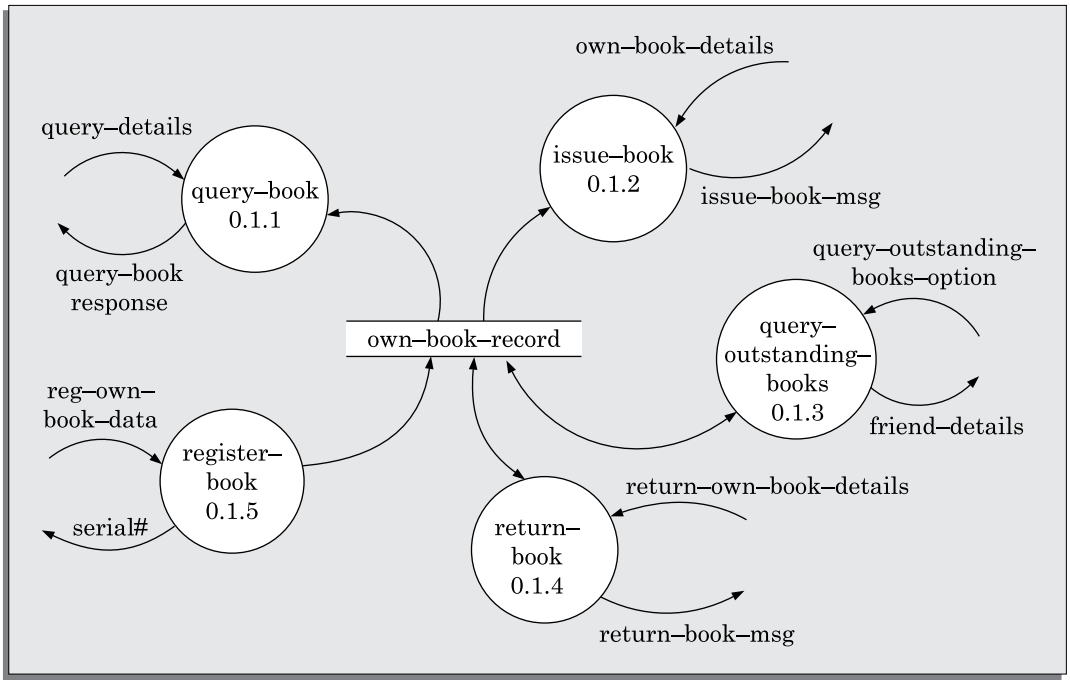


FIGURE 6.17 Level 2 DFD for Example 6.5.

Data dictionary for the DFD model of Example 6.5

input-data: friend-reg-data + own-book-data + stat-request + borrowed-book-data
response: friend-reg-conf-msg + own-book-response + stat-response +
borrowed-book-response
own-book-data: query-details + own-book-details + query-outstanding-books-option +
return-own-book-details + reg-own-book-data
own-book-response: query-book-response + issue-book-msg + friend-details + return-book-msg + serial#.
borrowed-book-data: borrowed-book-details + book-return-details + display-books-option
borrowed-book-response: reg-msg + unreg-msg + borrowed-books-list
friend-reg-data: name + address + landline# + mobile#
own-book-details: friend-reg-data + book-title + data-of-issue
return-own-book-details: book-title + date-of-return
friend-details: name + address + landline# + mobile# + book-list
borrowed-book-details: book-title + borrow-date
serial#: integer

Observation: Observe that since there are more than seven functional requirements for the personal library software, related requirements have been combined to have only five bubbles in the level 1 diagram. Only level 2 DFD has been shown, since the other DFDs are trivial and need not be drawn.

Shortcomings of the DFD model

DFD models suffer from several shortcomings. The important shortcomings of DFD models are the following:

- Imprecise DFDs leave ample scope to be imprecise. In the DFD model, we judge the function performed by a bubble from its label. However, a short label may not capture the entire functionality of a bubble. For example, a bubble named find-book-position has only intuitive meaning and does not specify several things, e.g. what happens when some input information is missing or is incorrect. Further, the find-book-position bubble may not convey anything regarding what happens when the required book is missing.
- Not-well defined control aspects are not defined by a DFD. For instance, the order in which inputs are consumed and outputs are produced by a bubble is not specified. A DFD model does not specify the order in which the different bubbles are executed. Representation of such aspects is very important for modelling real-time systems.
- Decomposition: The method of carrying out decomposition to arrive at the successive levels and the ultimate level to which decomposition is carried out are highly subjective and depend on the choice and judgment of the analyst. Due to this reason, even for the same problem, several alternative DFD representations are possible. Further, many times it is not possible to say which DFD representation is superior or preferable to another one.
- Improper data flow diagram: The data flow diagramming technique does not provide any specific guidance as to how exactly to decompose a given function into its subfunctions and we have to use subjective judgment to carry out decomposition.

6.3.3 Extending DFD Technique to make it Applicable to Real-time Systems

In a real-time system, some of the high-level functions are associated with deadlines. Therefore, a function must not only produce correct results but also should produce them by some prespecified time. For real-time systems, execution time is an important consideration for arriving at a correct design. Therefore, explicit representation of control and event flow aspects are essential. One of the widely accepted techniques for extending the DFD technique to real-time system analysis is the Ward and Mellor technique [1985]. In the Ward and Mellor notation, a type of process that handles only control flows is introduced. These processes representing control processing are denoted using dashed bubbles. Control flows are shown using dashed lines/arrows.

Unlike Ward and Mellor, Hatley and Pirbhai [1987] show the dashed and solid representations on separate diagrams. To be able to separate the data processing and the control processing aspects, a *control flow diagram* (CFD) is defined. This reduces the complexity of the diagrams. In order to link the data processing and control processing diagrams, a notational reference (solid bar) to a control specification is used. The CSPEC describes the following:

- The effect of an external event or control signal.
- The processes that are invoked as a consequence of an event.

Control specifications represents the behaviour of the system in two different ways:

- It contains a *state transition diagram* (STD). The STD is a sequential specification of behaviour.
- It contains a *program activation table* (PAT). The PAT is a combinatorial specification of behaviour. PAT represents invocation sequence of bubbles in a DFD.

6.4 STRUCTURED DESIGN

The aim of structured design is to transform the results of the structured analysis (that is, the DFD model) 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. Since 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 not represented.

The basic building blocks using which structure charts are designed are as following:

Rectangular boxes: A rectangular box represents a module. Usually, every rectangular box is annotated with the name of the module it represents.

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. However, just by looking at the structure chart, we cannot say whether a modules calls another module just once or many times. Also, just by looking at the structure chart, we cannot tell the order in which the different modules are invoked.

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 the arrow.

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 library module.

Selection: The diamond symbol represents the fact that one module of several modules connected with the diamond symbol is 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. 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 two higher-level modules to invoke the same lower-level module. An example of a properly layered design and another of a poorly layered design are shown in Figure 6.18.

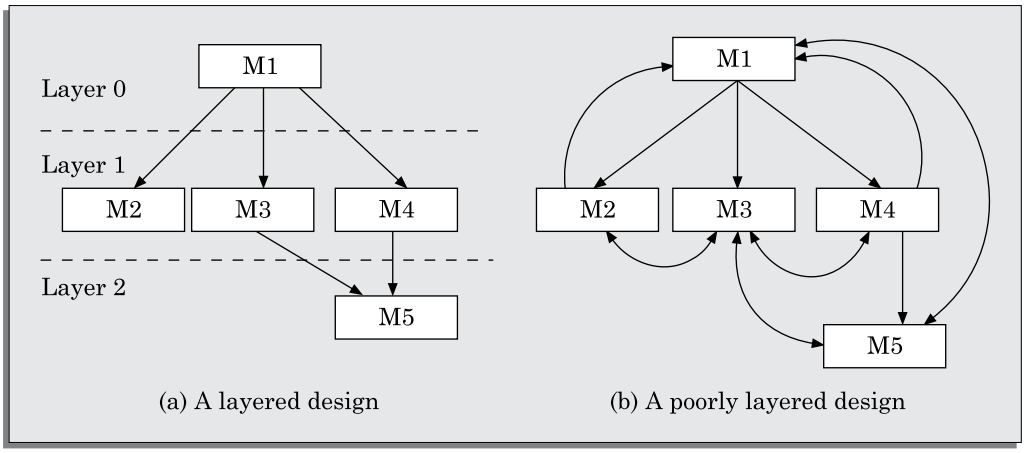


FIGURE 6.18 Examples of properly and poorly layered designs.

Flow chart *versus* structure chart

We are all familiar with the flow chart representation of a program. 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:

- It is usually difficult to identify the different modules of a program from its flow chart representation.
- Data interchange among different modules is not represented in a flow chart.
- Sequential ordering of tasks that is inherent to a flow chart is suppressed in a structure chart.

6.4.1 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 structure chart:

- Transform analysis
- Transaction analysis

At each level of transformation, it is important to first determine whether the transform or the transaction analysis is applicable to a particular DFD.

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.

Whether to apply transform or transaction processing?

Given a specific DFD of a model, how does one decide whether to apply transform analysis or transaction analysis? 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 dangling arrows. 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 simple processing.

Please recollect that the bubbles are decomposed until it represents a very simple processing that can be implemented using only a few lines of code. 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 these components. The first step in transform analysis is to divide the DFD into three types of parts:

- 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.*

The output portion of a DFD transforms output data from logical form to physical form. Each output portion is called an *efferent branch.* The remaining portion of a DFD is called *central transform*.

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.

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 central tansforms. The first level of structure chart is produced by representing each input and output unit as a box and each central transform as a single box.

In the third step of transform analysis, the structure chart is refined by adding subfunctions required by each of the high-level functional components. 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, initialisation and termination process, identifying consumer modules etc. The factoring process is continued until all bubbles in the DFD are represented in the structure chart.

PROBLEM 6.1 Draw the structure chart for the RMS software of Example 6.1.

Solution: By observing the level 1 DFD of Figure 6.8, 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 Figure 6.19.

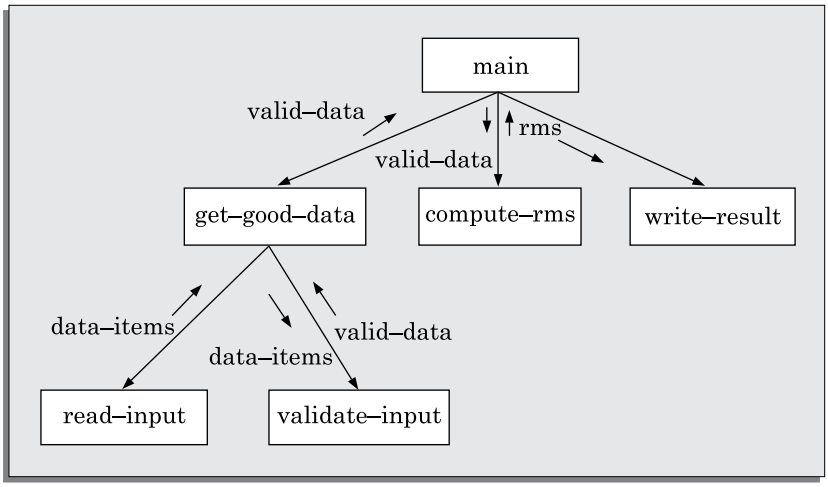


FIGURE 6.19 Structure chart for Problem 6.1.

PROBLEM 6.2 Draw the structure chart for the tic-tac-toe software of Example 6.2.

Solution: The structure chart for the Tic-tac-toe software is shown in Figure 6.20. Observe that the check-game-status bubble, though produces some outputs, is not really responsible for converting logical data to physical data. On the other hand, it carries out the processing involving checking game status. That is the main reason, why we have considered it as a central transform and not as an efferent type of module.

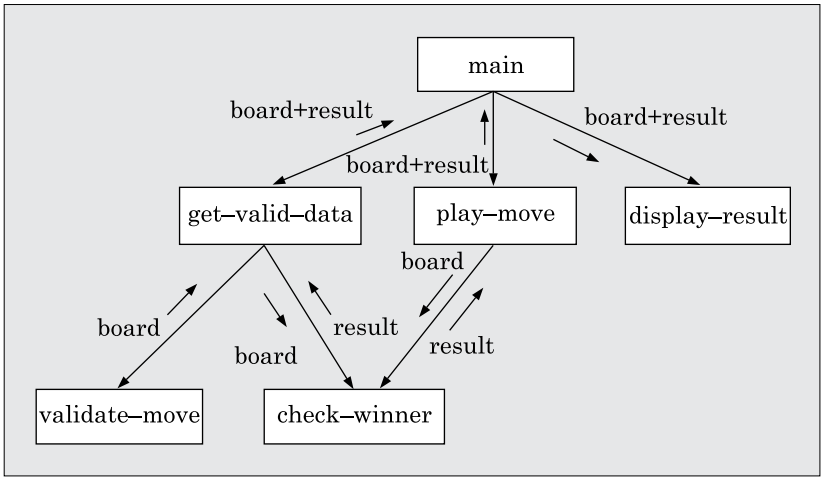


FIGURE 6.20 Structure chart for Problem 6.2.

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. A simple way to identify a transaction is the following. Check the input data. The number of bubbles on which the input data to the DFD are incident defines the number of transactions. However, some transactions may not require any input data. These transactions can be identified based on the experience gained from solving a large number of examples.

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. Every transaction carries a tag identifying its type. Transaction analysis uses this tag to divide the system into transaction modules and a transaction-center module.

PROBLEM 6.3 Draw the structure chart for the Supermarket Prize Scheme software of Example 6.3.

Solution: The structure chart for the Supermarket Prize Scheme software is shown in Figure 6.21.

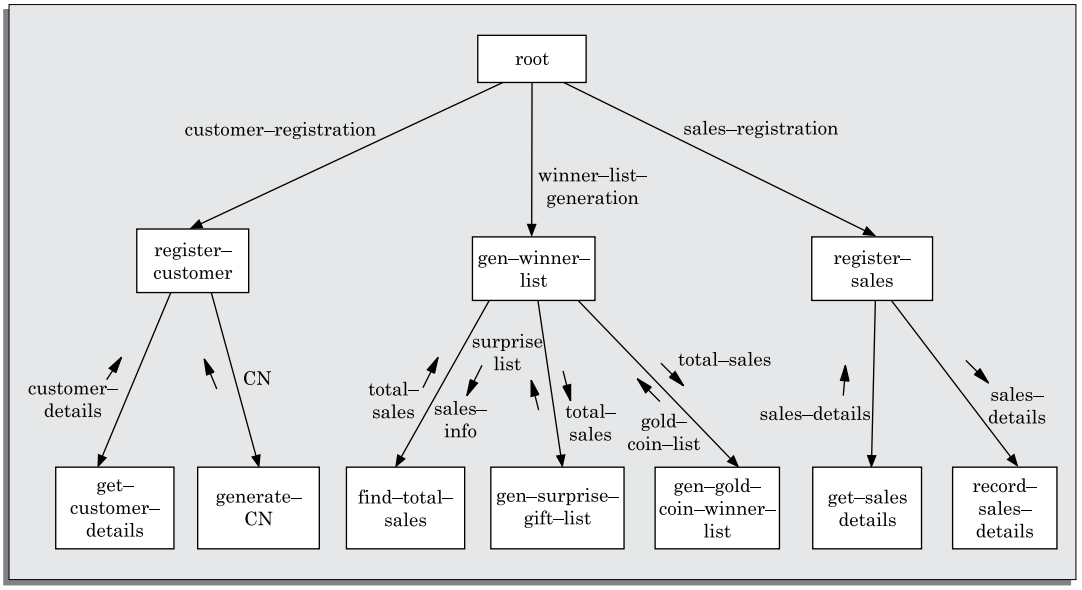


FIGURE 6.21 Structure chart for Problem 6.3.

PROBLEM 6.4 Draw the structure chart for the *trade-house automation system* (TAS) software of Example 6.4.

Solution: The structure chart for the *trade-house automation system* (TAS) software of Example 6.4 is shown in Figure 6.22.

By observing the level 1 DFD of Figure 6.14, we can see that the data input to the diagram are handled by different bubbles and therefore transaction analysis is applicable to this DFD. Input data to this DFD are handled in three different ways (accept-order, accept-indent-request, and handle-query), we have three different transactions corresponding to these as shown in Figure 6.22.

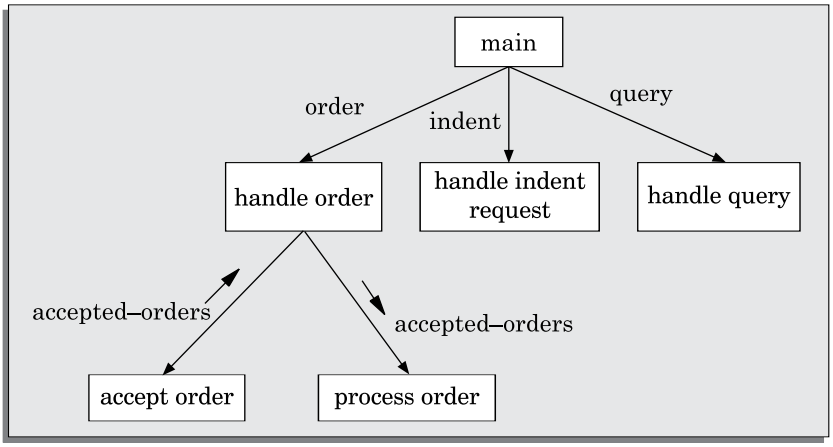


FIGURE 6.22 Structure chart for Problem 6.4.

Word of caution

We should view transform and transaction analyses as guidelines, rather than rules. We should apply these guidelines in the context of the problem and handle the pathogenic cases carefully.

PROBLEM 6.5 Draw the structure chart for the personal library software of Example 6.5.

Solution: The structure chart for the personal library software is shown in Figure 6.23.

6.5 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.

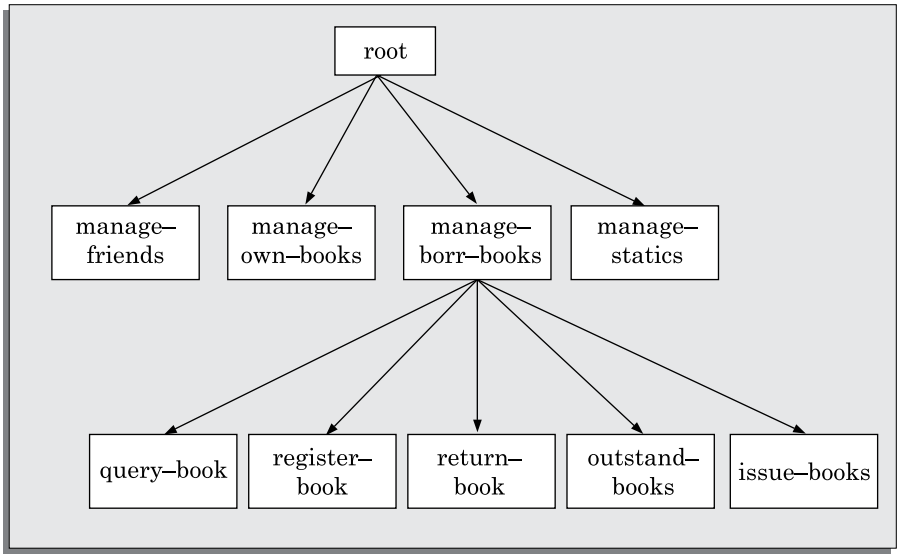


FIGURE 6.23 Structure chart for Problem 6.5.

6.6 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 development team. 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 following aspects:

Traceability: Whether each bubble of the DFD can be traced to some module in the structure chart and *vice versa*. They check whether each functional requirement in the SRS document can be traced to some bubble in the DFD model 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 be implemented.

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

SUMMARY

- In this chapter, we discussed a sample function-oriented software design methodology called structured analysis/structured design (SA/SD) which incorporates features of some important design methodologies.
- Methodologies like SA/SD give us a recipe for developing a good design according to the different goodness criteria we had discussed in Chapter 5.

- item SA/SD consists of two important parts—structured analysis and structured design.
- The goal of structured analysis is to perform a functional decomposition of the system. Results of structured analysis is represented using data flow diagrams (DFDs). The DFD representation is difficult to implement using a traditional programming language. The DFD representation can be systematically be transformed to structure chart representation. The structure chart representation can be easily implemented using a conventional programming language.
- During structured design, the DFD representation obtained during structured analysis is transformed into a structure chart representation.
- Several CASE tools are available to support the software design process carried out using the important function-oriented design methodologies. In addition to laying out the DFDs, structure charts, maintaining the data dictionary, and helping in traceability analysis, these CASE tools can also perform some elementary consistency checking, e.g., they can usually check whether a DFD is balanced or not.

EXERCISES

MULTIPLE CHOICE QUESTIONS

For each of the following questions, only one of the options is correct. Choose the correct option:

1. A data flow diagram represents:
(a) The conditions based on which a data may be processed
(b) The order in which different activities are carried out
(c) The transformation of data through processing stations
(d) The order in which various functions of a program are invoked

2. A DFD depicts which of the following:
(a) Flow of data
(b) Flow of control
(c) Flow of statements
(d) None of the above

3. Which of the following statements is not true of data flow diagrams (DFDs)?
(a) Hierarchical diagram.
(b) Represent code structure
(c) Do not represent decisions and control flows
(d) Represent functional decomposition

4. In a procedural design approach, during the detailed design stage, which of the following is undertaken?
(a) Module structure is designed
(b) Data flow representation is developed
(c) Data structures and algorithms for the individual modules are developed
(d) Structure chart is developed

REVIEW QUESTIONS

1. What do you understand by the term "top-down decomposition" in the context of function-oriented design?
2. Distinguish between a data flow diagram (DFD) and a flow chart.
3. Differentiate between structured analysis and structured design in the context of function-oriented design.
4. Point out the important differences between a structure chart and a flow chart as design representation techniques.
5. What do you mean by the term data dictionary in the context of structured analysis? How is a data dictionary useful during software development and maintenance?
6. Construct the DFD representation for the following program:

```
main(){                    .    f(int a,int b){      .    f1(){
   int a[100];             .      if (a>b) f1();     .       return;}
   for(i=0;i<100;i++)      .      else f2();         .
      f(a[i],a[i+1]);      .    }                    .    f2(){
}                          .                         .       return;}
```

7. Explain how a DFD model of software can be created from its source code.
8. What do you understand by the terms "structured analysis" and "structured design"? What are the main objectives of "structured analysis" and "structured design"?
9. Explain how the DFD model can help one understand the working of a software system.
10. State whether the following statement is true or false. "The essence of any good function-oriented design principle is to map similar functions into a module." Give reasons behind your answers.
11. Identify the correct statement. Give reasoning behind your choice.
    (a) A DFD model essentially represents the data and control relationships among program elements.
    (b) A DFD model of a system usually comprises many DFDs.
    (c) The DFD model is the design model of a system.
    (d) A DFD model cannot represent a system's file data storage.
12. What do you mean by balancing a DFD? Illustrate your answer with a suitable example.
13. What are the main shortcomings of data flow diagram (DFD) as a tool for performing structured analysis?
14. Why is design reviews important? Suppose you are required to review a SA/SD document, make a list of items that can be used as a checklist for carrying out the review.

15. What do you understand by design review? What kinds of mistakes are normally pointed out by the reviewers?

16. (a) Draw a labelled DFD for the following time management software. Clearly show the context diagram and its hierarchical decompositions up to level 2. (*Note:* Context diagram is the Level 0 DFD).

A company needs to develop a time management system for its executives. The software should let the executives register their daily appointment schedules. The information to be stored includes person(s) with whom meeting is arranged, venue, the time and duration of the meeting, and the purpose (e.g., for a specific project work). When a meeting involving many executives needs to be organised, the system should automatically find a common slot in the diaries of the concerned executives, and arrange a meeting (i.e., make relevant entries in the diaries of all the concerned executives) at that time. It should also inform the concerned executives about the scheduled meeting through e-mail. If no common slot is available, TMS should help the secretary to rearrange the appointments of the executives in consultation with the concerned executives for making room for a common slot. To help the executives check their schedules for a particular day the system should have a very easy-to-use graphical interface. Since the executives and the secretaries have their own desktop computers, the time management software should be able to serve several remote requests simultaneously. Many of the executives are relative novices in computer usage. Everyday morning the time management software should e-mail every executive his appointments for the day. Besides registering their appointments and meetings, the executives might mark periods for which they plan to be on leave. Also, executives might plan out the important jobs they need to do on any day at different hours and post it in their daily list of engagements. Other features to be supported by the TMS are the following—TMS should be able to provide several types of statistics such as which executive spent how much time on meetings. For which project how many meetings were organised for what duration and how many man-hours were devoted to it. Also, it should be able to display for any given period of time the fraction of time that on the average each executive spent on meetings.

(b) Using the DFD you have developed for Part (a) of this question, develop the structured design for the time management software.

17. A hotel has a certain number of rooms. Each room can be either single bed or double bed type and may be AC or non-AC type. The rooms have different rates depending on whether they are of single or double, AC or Non-AC types. The room tariff however may vary during different parts of the year depending up on the occupancy rate. For this, the computer should be able to display the average occupancy rate for a given month, so that the manager can revise the room tariff for the next month either upwards or downwards by a certain percentage. Perform structured analysis and structured design for this **Hotel Automation Software**—software that would automate the book keeping activities of a 5-star hotel.

Guests can reserve rooms in advance or can reserve rooms on the spot depending upon availability of rooms. The receptionist would enter data pertaining to guests such as their arrival time, advance paid, approximate duration of stay, and the type

of the room required. Depending on this data and subject to the availability of a suitable room, the computer would allot a room number to the guest and assign a unique token number to each guest. If the guest cannot be accommodated, the computer generates an apology message. The hotel catering services manager would input the quantity and type of food items as and when consumed by the guest, the token number of the guest, and the corresponding date and time. When a customer prepares to check-out, the hotel automation software should generate the entire bill for the customer and also print the balance amount payable by him. During check-out, guests can opt to register themselves for a frequent guests programme. Frequent guests should be issued an identity number which helps them to get special discounts on their bills.

18. Perform structured analysis and structured design (SA/SD) for a software to be developed for automating various book keeping activities of a small book shop. From a discussion with the owner of the book shop, the following user requirements for this **Book-shop Automation Software (BAS)**—have been arrived at:

BAS should help the customers query whether a book is in stock. The users can query the availability of a book either by using the book title or by using the name of the author. If the book is not currently being sold by the book-shop, then the customer is asked to enter full details of the book for procurement of the book in future. The customer can also provide his e-mail address, so that he can be intimated automatically by the software as and when the book copies are received. If a book is in stock, the exact number of copies available and the rack number in which the book is located should be displayed. If a book is not in stock, the query for the book is used to increment a request field for the book. The manager can periodically view the request field of the books to arrive at a rough estimate regarding the current demand for different books. BAS should maintain the price of various books. As soon as a customer selects his books for purchase, the sales clerk would enter the ISBN numbers of the books. BAS should update the stock, and generate the sales receipt for the book. BAS should allow employees to update the inventory whenever new supply arrives. Also upon request by the owner of the book shop, BAS should generate sales statistics (viz., book name, publisher, ISBN number, number of copies sold, and the sales revenue) for any period. The sales statistics will help the owner to know the exact business done over any period of time and also to determine inventory level required for various books. The inventory level required for a book is equal to the number of copies of the book sold over a period of one week multiplied by the average number of weeks it takes to procure the book from its publisher. Every day the book shop owner would give a command for the BAS to print the books which have fallen below the threshold and the number of copies to be procured along with the full address of the publisher.

19. Perform structured analysis and structured design for the following **City Corporation Automation Software (CCAS)** to be developed for automating various book keeping activities associated with various responsibilities of the Municipal Corporation of a large city.

A city corporation wishes to develop a web-site using which the residents can get information on various facilities being provided by the corporate to the citizens. Since

the city population exceeds 5 lakh, the maximum number of concurrent clicks can be upto 10 clicks per second. The corporation also plans to use the same web site for its road maintenance activity.

A city corporation has branch offices at different suburbs of the city. Residents would raise repair requests for different roads of the city on line. The supervisor at each branch office should be able to view all new repair requests pertaining to his area. Soon after a repair request is raised, a supervisor visits the road and studies the severity of road condition. Depending on the severity of the road condition and the type of the locality (e.g., commercial area, busy area, relatively deserted area, etc.), he determines the priority for carrying out this repair work. The supervisor also estimates the raw material requirement for carrying out the repair work, the types and number of machines required, and the number and types of personnel required. The supervisor enters this information through a special login in the web site. Based on this data, the system should schedule the repair of the road depending on the priority of the repair work and subject to the availability of raw material, machines, and personnel. This schedule report is used by the supervisor to direct different repair work. The manpower and machine that are available are entered by the city corporation administrator. He can change these data any time. Of course, any change to the available manpower and machine would require a reschedule of the projects. The progress of the work is entered periodically by the supervisor which can be seen by the citizens in the web site.

The mayor of the city can request for various road repair statistics such as the number and type of repairs carried out over a period of time and the repair work outstanding at any point of time and the utilisation statistics of the repair manpower and machine over any given period of time.

20. Perform structured analysis and structured design for developing the following **Restaurant Automation System** using the SA/SD technique.

A restaurant owner wants to computerise his order processing, billing, and accounting activities. He also expects the computer to generate statistical report about sales of different items. A major goal of this computerisation is to make supply ordering more accurate so that the problem of excess inventory is avoided as well as the problem of non-availability of ingredients required to satisfy orders for some popular items is minimised. The computer should maintain the prices of all the items and also support changing the prices by the manager. Whenever any item is sold, the sales clerk would enter the item code and the quantity sold. The computer should generate bills whenever food items are sold. Whenever ingredients are issued for preparation of food items, the data is to be entered into the computer. Purchase orders are generated on a daily basis, whenever the stock for any ingredient falls below a threshold value. The computer should calculate the threshold value for each item based on the average consumption of this ingredient for the past three days and assuming that a minimum of two days stock must be maintained for all ingredients. Whenever the ordered ingredients arrive, the invoice data regarding the quantity and price is entered. If sufficient cash balance is available, the computer should print cheques immediately against invoice. Monthly sales receipt and expenses data should be generated whenever the manager would request to see them.

21. Perform structured analysis and design for the following **Judiciary Information System (JIS) software.**

The attorney general's office has requested us to develop a Judiciary Information System (JIS), to help handle court cases and also to make the past court cases easily accessible to the lawyers and judges. For each court case, the name of the defendant, defendant's address, the crime type (e.g., theft, arson, etc.), when committed (date), where committed (location), name of the arresting officer, and the date of the arrest are entered by the court registrar. Each court case is identified by a unique *case identification number* (CIN) which is generated by the computer. The registrar assigns a date of hearing for each case. For this the registrar expects the computer to display the vacant slots on any working day during which the case can be scheduled. Each time a case is adjourned, the reason for adjournment is entered by the registrar and he assigns a new hearing date. If hearing takes place on any day for a case, the registrar enters the summary of the court proceedings and assigns a new hearing date. Also, on completion of a court case, the summary of the judgment is recorded and the case is closed but the details of the case is maintained for future reference. Other data maintained about a case include the name of the presiding judge, the public prosecutor, the starting date, and the expected completion date of a trial. The judges should be able to browse through the old cases for guidance on their judgment. The lawyers should also be permitted to browse old cases, but should be charged for each old case they browse. Using the JIS software, the Registrar of the court should be able to query the following:

(a) The currently pending court cases. In response to this query, the computer should print out the pending cases sorted by CIN. For each pending case, the following data should be listed—the date in which the case started, the defendant's name, address, crime details, the lawyer's name, the public prosecutor's name, and the attending judge's name.

(b) The cases that have been resolved over any given period. The output in this case should chronologically list the starting date of the case, the CIN, the date on which the judgment was delivered, the name of the attending judge, and the judgment summary.

(c) The cases that are coming up for hearing on a particular date.

(d) The status of any particular case (cases are identified by CIN).

22. The different activities of the library of our institute pertaining to the issue and return of the books by the members of the library and various queries regarding books as listed below are to be automated. Perform structured analysis and structured design for this Library Information System (LIS) software:

- The library has 10,000 books. Each book is assigned a unique identification number (called ISBN number). The Library clerk should be able to enter the details of the book into the LIS through a suitable interface.
- There are four categories of members of the library—undergraduate students, post graduate students, research scholars, and faculty members.
- Each library member is assigned a unique library membership code number.
- Each undergraduate student can issue up to 2 books for 1 month duration.
- Each postgraduate student can issue up to 4 books for 1 month duration.

- Each research scholar can issue up to 6 books for 3 months duration.
- Each faculty member can issue up to 10 books for six months duration.
- The LIS should answer user queries regarding whether a particular book is available. If a book is available, LIS should display the rack number in which the book is available and the number of copies available.
- LIS registers each book issued to a member. When a member returns a book, LIS deletes the book from the member's account and makes the book available for future issue.
- Members should be allowed to reserve books which have been issued. When such a reserved book is returned, LIS should print a slip for the concerned member to get the book issued and should disallow issue of the book to any other member for a period of seven days or until the member who has reserved the books gets it issued.
- When a member returns a book, LIS prints a bill for the penalty charge for overdue books. LIS calculates the penalty charge by multiplying the number of days the book is overdue by the penalty rate.
- LIS prints reminder messages for the members against whom books are overdue, upon a request by the Librarian.
- LIS should allow the Librarian to create and delete member records. Each member should be allocated a unique membership identification number which the member can use to issue, return, and reserve books.

23. Perform the SA/SD for the following word processing software.

- The word processing software should be able to read text from an ASCII file or HTML file and store the formatted text as HTML files in the disk.
- The word processing software should ask the user about the number of characters in an output line of the formatted text. The user should be allowed to select any number between 1 and 132.
- The word processing software should process the input text in the following way.
  - Each output line is to contain exactly the number of characters specified by the user (including blanks).
  - The word processing software is to both left and right justify the text so that there are no blanks at the left- and right-hand ends of lines except the first and possibly the last lines of paragraphs. The word processing software should do this by inserting extra blanks between words.
  - The input text from the ASCII file should consist of words separated by one or more blanks and a special character PP, which denotes the end of a paragraph and the beginning of another.
  - The first line of each paragraph should be indented by five spaces and should be right justified.
  - The last line of each paragraph should be left justified.
- The user should be able to browse through the document and add, modify or delete words. He/she should also be able to mark any word as bold, italic, superscript, or subscript.

- The user can request to see the number of characters, words, lines, and paragraphs used in the document.
- The user should be able to save his documents under a name specified by him.

24. It is required to develop a graphics editor software package using which one can create/modify several types of graphics entities. In summary, the graphics editor should support the following features: (Those who are not familiar with any graphics editor, please look at the Graphics Drawing features available in either MS-Word or PowerPoint software. You can also examine any other Graphical Drawing package accessible to you. An understanding of the standard features of a Graphics Editor will help you understand the different features required.)

- The graphics editor should support creating several types of geometric objects such as circles, ellipses, rectangles, lines, text, and polygons.
- Any created object can be *selected* by clicking a mouse button on the object. A selected object should be shown in a highlighted color.
- A selected object can be edited, i.e., its associated characteristics such as its geometric shape, location, color, fill style, line width, line style, etc. can be changed. For texts, the text content can be changed.
- A selected object can be copied, moved, or deleted.
- The graphics editor should allow the user to save his created drawings on the disk under a name he would specify. The graphics editor should also support loading previously created drawings from the disk.
- The user should be able to define any rectangular area on the screen to be *zoomed* to fill the entire screen.
- A *fit screen function* makes the entire drawing fit the screen by automatically adjusting the zoom and pan values.
- A pan function should allow the displayed drawing to be panned along any direction by a specified amount.
- The graphics editor should support grouping. A group is simply a set of drawing objects including other groups which when grouped behave as a single entity. This feature is especially useful when you wish to manipulate several entities in the same way. A drawing object can be a direct member of at most one group. It should be possible to perform several editing operations on a group such as move, delete, and copy.
- A set of 10 clip boards should be provided to which one can copy various types of selected entities (including groups) for future use in pasting these at different places when required.

25. Perform SA/SD for the following **Software component cataloguing software.**

Software component cataloguing software: The software component cataloguing software consists of a software components catalogue and various functions defined on this components catalogue. The software components catalogue should hold details of the components which are potentially reusable. The reusable components can be either design or code. The design might have been constructed using different design notations such as UML, ERD, structured design, etc. Similarly, the code might have been written using different programming languages. A cataloguer may enter

components in the catalogue, may delete components from the catalogue, and may associate reuse information with a catalogue component in the form of a set of key words. A user of the catalogue may query about the availability of a component using certain key words to describe the component. In order to help manage the component catalogue (i.e., periodically purge the unused components) the cataloguing software should maintain information such as how many times a component has been used, and how many times the component has come up in a query but not used. Since the number of components usually tend to be very high, it is desirable to be able to classify the different types of components hierarchically. A user should be able to browse the components in each category.

26. The manager of a supermarket wants us to develop an automation software. The supermarket stocks a set of items. Customers pick up their desired items from the different counters in required quantities. The customers present these items to the sales clerk. The sales clerk enters the code number of these items along with the respective quantity/units. Perform structured analysis and structured design for developing this **Supermarket Automation Software (SAS).**

- SAS should at the end of a sales transaction print the bill containing the serial number of the sales transaction, the name of the item, code number, quantity, unit price, and item price. The bill should indicate the total amount payable.
- SAS should maintain the inventory of the various items of the supermarket. The manager upon query should be able to see the inventory details. In order to support inventory management, the inventory of an item should be decreased whenever an item is sold. SAS should also support an option by which an employee can update the inventory when new supply arrives.
- SAS should support printing the sales statistics for every item the supermarket deals with for any particular day or any particular period. The sales statistics should indicate the quantity of an item sold, the price realised, and the profit.
- The manager of the supermarket should be able to change the price at which an item is sold as the prices of the different items vary on a day-to-day basis.

27. A transport company wishes to computerise various book keeping activities associated with its operations. Perform structured analysis and structured design for developing the **Transport Company Computerisation (TCC) software:**

- A transport company owns a number of trucks.
- The transport company has its head office located at the capital and has branch offices at several other cities.
- The transport company receives consignments of various sizes at (measured in cubic metres) its different offices to be forwarded to different branch offices across the country.
- Once the consignment arrives at the office of the transport company, the details of the volume, destination address, sender address, etc., are entered into the computer. The computer would compute the transport charge depending upon the volume of the consignment and its destination and would issue a bill for the consignment.
- Once the volume of any particular destination becomes 500 m^3, the Computerisation system should automatically allot the next available truck.

- A truck stays with the branch office until the branch office has enough cargo to load the truck fully.
- The manager should be able to view the status of different trucks at any time.
- The manager should be able to view truck usage over a given period of time.
- When a truck is available and the required consignment is available for dispatch, the computer system should print the details of the consignment number, volume, sender's name and address, and the receiver's name and address to be forwarded along with the truck.
- The manager of the transport company can query the status of any particular consignment and the details of volume of consignments handled to any particular destination and the corresponding revenue generated.
- The manager should also be able to view the average waiting period for different consignments. This statistics is important for him since he normally orders new trucks when the average waiting period for consignments becomes high due to non-availability of trucks. Also, the manager would like to see the average idle time of the truck in the branch for a given period for future planning.

28. Draw level 0 (context level) and level 1 data flow diagram for the following students' academic record management software.

- A set of courses are created. Each course consists of a unique course number, number of credits, and the syllabus.
- Students are admitted to courses. Each students' details include his roll number, address, semester number and the courses registered for the semester.
- The marks of student for various units he credited are keyed in.
- Once the marks are keyed in, the semester weighted average (SWA) is calculated.
- The recent marks of the student are added to his previous marks and a weighted average based on the credit points for various units is calculated.
- The marks for the current semester are formatted and printed.
- The SWA appears on the report.
- A check must be made to determine if a student should be placed on the Vice-Chancellor's list. This is determined based on whether a student scores an SWA of 85 or higher.
- If the SWA is lower than 50, the student is placed on a conditional standing.

29. Perform structured analysis and structured design (SA/SD) for the following **CASE tool for Structured Analysis Software** to be developed for automating various activities associated with developing a CASE tool for structured software analysis.

- The case tool should support a graphical interface and the following features.
- The user should be able to draw bubbles, data stores, and entities and connect them using data flow arrows. The data flow arrows are annotated by the corresponding data names.
- Should support editing the data flow diagram.
- Should be able to create the diagram hierarchically.
- The user should be able to determine balancing errors whenever required.
- The software should be able to create the data dictionary automatically.

- Should support printing the diagram on a variety of printers.
- Should support querying the data items and function names. The diagrams matching the query should be shown.

30. Perform structured analysis and structured design (SA/SD) for a software to be developed for automating various activities associated with developing a CASE tool for structured software design. The summary of the requirements for this **CASE tool for Structured Design** are the following:

- The case tool should support a graphical interface and the following features.
- It should be possible to import the DFD model developed by another program. The user should be able to apply the transform and transaction analysis to the imported DFD model.
- The user should be able to draw modules, control arrows, and data flow arrows. Also symbol for library modules should be provided. The data flow arrows are annotated with the corresponding data name.
- The modules should be organised in hierarchical levels.
- The user should be able to modify his design. Please note that when he deletes a data flow arrow, its annotated data name automatically gets deleted.
- For large software, modules may be hierarchically organised and clicking on a module should be able to show its internal organisation.
- The user should be able to save his design and also be able to load previously created designs.

31. The local newspaper and magazine delivery agency has asked us to develop a software for him to automate various clerical activities associated with his business. Perform the structured analysis and design for this **Newspaper Agency Automation Software.**

- This software is to be used by the manager of the news agency and his delivery persons.
- For each delivery person, the system must print each day the publications to be delivered to each address.
- The customers usually subscribe one or more newspapers and magazines. They are allowed to change their subscription notice by giving one week's advance notice. Customers should be able to initiate new subscriptions and suspend subscription for a particular item either temporarily or permanently through a web browser. Considering the large customer base, at least 10 concurrent customer accesses should be supported.
- For each delivery person, the system must print each day the publications to be delivered to each address.
- The system should also print for the news agent the information regarding who received what and a summary information of the current month.
- At the beginning of every month bills are printed by the system to be delivered to the customers. These bills should be computed by the system automatically.
- The customers may ask for stopping the deliveries to them for certain periods when they go out of station.

- Customers may request to subscribe new newspapers/magazines, modify their subscription list, or stop their subscription altogether.
- Customers usually pay their monthly dues either by cheque or cash. Once the cheque number or cash received is entered in the system, receipt for the customer should be printed.
- If any customer has any outstanding due for one month, a polite reminder message is printed for him and his subscription is discontinued if his dues remain outstanding for periods of more than two months.
- The software should compute and print out the amount payable to each delivery boy. Each delivery boy gets 2.5 per cent of the value of the publications delivered by him.

32. Perform SA/SD for the following **University Department Information System.** This software concerns automating the activities of department offices of universities. Department offices in different universities do a lot of book-keeping activities the software to be developed targets to automate these activities.

- Various details regarding each student such as his name, address, course registered, etc. are entered at the time he takes admission.
- At the beginning of every semester, students do course registration. The information system should allow the department secretary to enter data regarding student course registrations. As the secretary enters the roll number of each student, the computer system should bring up a form for the corresponding student and should keep track of courses he has already completed and the courses he has back-log, etc.
- At the end of the semester, the instructors leave their grading information at the office which the secretary enter in the computer. The information system should be able to compute the grade point average for each student for the semester and his *cumulative grade point average* (CGPA) and print the grade sheet for each student.
- The information system also keeps track of inventories of the Department, such as equipment, their location, furniture, etc.
- The Department has a yearly grant and the Department spends it in buying equipment, books, stationery items, etc. Also, in addition to the annual grant that the Department gets from the University, it gets funds from different consultancy service it provides to different organisations. It is necessary that the Department information system keeps track of the Department accounts.
- The information system should also keep track of the research projects of the Department, publications by the faculties, etc. These information are keyed in by the secretary of the Department.
- The information system should support querying the up-to-date details about every student by inputting his roll number. It should also support querying the details of the cash book account. The output of this query should include the income, expenditure, and balance.

33. Perform SA/SD to develop a software to automate the activities of a small automobile spare parts shop. The small automobile spare parts shop sells the spare parts for a vehicles of several makes and models. Also, each spare part is typically manufactured by several small industries. To stream line the sales and supply ordering, the shop

owner has asked us to develop the following motor part shop software. Perform the SA/SD for this **Motor Part Shop Software (MPSS).**

The motor parts shop deals with large number of motor parts of various manufacturers and various vehicle types. Some of the motor parts are very small and some are of reasonably large size. The shop owner maintains different parts in wall mounted and numbered racks.

The shop owner maintains as few inventory for each item as reasonable, to reduce inventory overheads after being inspired by the just-in-time (JIT) philosophy.

Thus, one important problem the shop owner faces is to be able to order items as soon as the number of items in the inventory reduces below a threshold value. The shop owner wants to maintain parts to be able to sustain selling for about one week. To calculate the threshold value for each item, the software must be able to calculate the average number of parts sales for one week for each part.

At the end of each day, the shop owner would request the computer to generate the items to be ordered. The computer should print out the part number, the amount required and the address of the vendor supplying the part.

The computer should also generate the revenue for each day and at the end of the month, the computer should generate a graph showing the sales for each day of the month.

34. Perform structured analysis and structured design for the following **Medicine Shop Automation (MSA)** software:

A retail medicine shop deals with a large number of medicines procured from various manufacturers. The shop owner maintains different medicines in wall mounted and numbered racks.

- The shop owner maintains as few inventory for each item as reasonable, to reduce inventory overheads after being inspired by the just-in-time (JIT) philosophy.
- Thus, one important problem the shop owner faces is to be able to order items as soon as the number of items in the inventory reduces below a threshold value. The shop owner wants to maintain medicines to be able to sustain selling for about one week. To calculate the threshold value for each item, the software must be able to calculate the average number of medicines sales for one week for each part.
- At the end of each day, the shop owner would request the computer to generate the items to be ordered. The computer should print out the medicine description, the quantity required, and the address of the vendor supplying the medicine. The shop owner should be able to store the name, address, and the code numbers of the medicines that each vendor deals with.
- Whenever new supply arrives, the shop owner would enter the item code number, quantity, batch number, expiry date, and the vendor number. The software should print out a cheque favouring the vendor for the items supplied.
- When the shop owner procures new medicines it had not dealt with earlier, he should be able to enter the details of the medicine such as the medicine trade name, generic name, vendors who can supply this medicine, unit selling

and purchasing price. The computer should generate a code number for this medicine which the shop owner would paste the code number in the rack where this medicine would be stored. The shop owner should be able to query about a medicine either using its generic name or the trade name and the software should display its code number and the quantity present.

- At the end of every day the shop owner would give a command to generate the list of medicines which have expired. It should also prepare a vendor-wise list of the expired items so that the shop owner can ask the vendor to replace these items. Currently, this activity alone takes a tremendous amount of labour on the part of the shop owner and is a major motivatour for the automation endeavour.
- Whenever any sales occurs, the shop owner would enter the code number of each medicine and the corresponding quantity sold. The MSA should print out the cash receipt.
- The computer should also generate the revenue and profit for any given period. It should also show vendor-wise payments for the period.

35. The IIT students' Hall Management Center (HMC) has requested us to develop the following software to automate various book-keeping activities associated with its day-to-day operations.

- After a student takes admission, he/she presents a note from the admission unit, along with his/her name, permanent address, contact telephone number, and a photograph. He/she is then allotted a hall, and also a specific room number. A letter indicating this allotted room is issued to the concerned student.
- Students incur mess charges every month. The mess manager would input to the software the total charges for each student in a month on mess account.
- Each room has a fixed room rent. The newly constructed halls have higher rent compared to some of the older halls. Twin sharing rooms have lower rent.
- Each hall provides certain amenities to the students such as reading rooms, play rooms, TV room, etc. A fixed amount is levied on each student on this count.
- The total amount collected from each student of a hall towards mess charges is handed over to the mess manager every month. For this, the computer needs to print a sheet with the total amount due to each mess manager is printed. Printed cheques are issued to each manager and signatures are obtained from them on the sheet.
- Whenever a student comes to pay his dues, his total due is computed as the sum of mess charge, amenity charge, and room rent.
- The students should be able to raise various types of complaints using a web browser in their room or in the Lab. The complaints can be repair requests such as fused lights, non-functional water taps, non-functional water filters, room repair, etc. They can also register complaints regarding the behaviour of attendants, mess staff, etc. For this round-the-clock operation of the software is required.
- The HMC receives an annual grant from the Institute for staff salary and the upkeep of the halls and gardens. The HMC chairman should be provided support for distribution of the grant among the different halls. The Wardens of different halls should be able to enter their expenditure details against the allocations.

- The controlling warden should be able to view the overall room occupancy.
- The warden of each hall should be able to find out the occupancy of his hall. He should also be able to view the complaints raised by the students and post his *action taken report* (ATR) to each complaint.
- The halls employ attendants and gardeners. These temporary employees receive a fixed pay on a per day basis. The Hall clerk enters any leave taken by an attendant or a gardener from at the terminal located at the hall office. At the end of every month a consolidated list of salary payable to each employee of the hall along with cheques for each employee is printed out.
- The HMC incurs petty expenses such as repair works carried out, newspaper and magazine subscriptions, etc. It should be possible to enter these expenses.
- Whenever a new staff is recruited his details including his daily pay is entered. Whenever a staff leaves, it should be possible to delete his records.
- The warden should be able to view the statement of accounts any time. The warden would take a print out of the annual consolidated statement of accounts, sign and submit it to the Institute administration for approval and audit verification.

36. **IIT security software:** The security office of IIT is in need of a software to control and monitor the vehicular traffic into and out of the campus. The functionalities required of the software are as follows—Each vehicle in the IIT campus would be registered with the system. For this, each of the faculty, staff, and students owning one or more vehicles would have to fill up a form at the security office detailing the vehicle registration numbers, models, and other relevant details for the vehicles that they own. These data would be entered into the computer by a security staff after a due diligence check.

    Each time a vehicle enters or leaves the campus, a camera mounted near the check gate would determine the registration number of an incoming (or outgoing) vehicle and the model of the vehicle and input into the system. If the vehicle is a campus vehicle, then the check gate should lift automatically to let it in or out, as the case may be. Various details regarding the entry and exit of a campus vehicle such as its number, owner, date and time of entry/exit would be stored in the database for statistical purposes. For each outside vehicle entering the campus, the driver would be required to fill-up a form detailing the purpose of entry. This information would immediately be entered by the security personnel at the gate and along with this information, the information obtained from the camera such as the vehicle's model number, registration number, and photograph would be stored in the database. When an outside vehicle leaves the campus, the exit details such as date and time of exit would automatically be registered in the database. For any external vehicle that remains inside the campus for more than 8 hours, the driver would be stopped by the security personnel manning the gate, queried to satisfaction, and the response would be entered into the system.

    Considering that there have been several incidents of speeding and rough driving in the past, the security personnel manning various traffic intersections and other sensitive points of the campus would be empowered to telephone the registration number of an errant vehicle to the main gate. The security personnel at the main gate would enter this information into the computer. The driver of an errant vehicle

would be quizzed at the check gate during exit and the vehicle's future entry would be barred if the response is not found to be satisfactory. For each errant campus vehicle, the driver would be quizzed during the next exit, and if the response is not found to be satisfactory, a letter should get issued to the dean (campus affairs) (detailing the date, time, traffic point at which the incident occurred, and the nature of the offence) to deal with the concerned staff or student, as the case might be.

The security officer should be able to view the statistics pertaining to the total number of vehicles going in and coming out of the campus (over a day, month or year) and the total number of vehicles currently inside the campus has left campus, and if it indeed has, its time of departure should be displayed. The security officer can query whether a particular vehicle is currently inside the campus. The security officer can also query the total number of vehicles owned by the residents of the campus. Since campus security is a critical operation, adequate safety against cyber-attacks on the security software should be ensured. Also, considering the criticality of the operation, down times of more than 5 minutes would normally not be unacceptable.

Implementation simplification: While writing code, the commands for the check gate need only be displayed on the terminal and the inputs from the camera can be simulated through keyboard entry.

37. **Courier company computerisation (CCC) software:** A courier company wishes to computerise various book keeping activities associated with its daily operation. The courier company has branches in most important towns in India. It is proposed that the different branch offices be equipped with a computer and printer each. The developed software would be deployed on the computer at each branch office and linked though the Internet. The other details are as follows:

- At each of its branch office and other retail outlets, the courier company receives consignments of various weights and sizes (measured in cubic metres). The charges are at present ₹5,000 per cubic metre for distances upto 500 km. For larger distances, 10 per cent additional charge is levied for every 100 km or part thereof. For packets weighing more than 100 kg/m^3, an additional 10 per cent levy is charged for every 20 kg/m^3. For small articles and letters, ₹50 per 100 g is charged. At present, only those packets having destination to a city where a branch office is located is accepted.
- When a customer tries to book a consignment at any of the retail points or branch offices, the details of the consignment such as its volume, weight, destination address, sender address, etc. are entered into the computer by the sales clerk. The computer would compute the charges for the consignment and print a bill indicating a unique id indicating the consignment number, which is assigned to the consignment. A customer should be able to track the delivery status of the consignment on-line by using the unique id.
- The courier company owns a number of trucks, which are used for transporting consignments between branch offices.
- When the volume of consignments for any particular destination (branch office) becomes 500 m^3, the system should automatically allot the next available truck that is present at the branch office. Since, no consignment should get unduly delayed,

whenever a consignment cannot be dispatched to its destination within 3 days of its receipt, it should automatically be forwarded to any branch office that is closer to the destination. When a truck is allocated for dispatch of consignments to a branch office, the computer system should print the details of the consignment number, volume, sender's name and address, and the receiver's name and address. This print out is to be carried by the truck driver for monitoring and excise clearance purposes.

- When a truck reaches a branch office, its arrival status is updated. A truck stays with the branch office until the branch office has enough cargo to load the truck to at least 80 per cent of its capacity. The transport office at the branch office can enter the fuel and repair charges for a truck.
- The regular expenses of the courier company includes staff salaries, rental charges for the branch offices, and truck maintenance charges. The company judiciously uses its profits to set up new branch offices and to buy additional trucks.
- The software should maintain the details of each employee such as his name, address, telephone number, basic pay, and other allowances. It should help the account manager at each branch office to generate the pay slip of all the employees every month and automatically credit the salaries to their respective bank accounts.
- All payments and receipts are entered into the system. A consolidated profit-loss account (taking all the branches and the entire operation into account) is expected to be prepared by the system. The manager should be able to view branch-wise revenue generated, consignments handled, expenses, etc.
- The manager should be able to view the status of different trucks at any time, e.g., the branch office at which it is waiting, or the two branch offices between which it is currently transporting consignments.
- The manager should be able to view truck usage (overall as well as for individual trucks) over a given period of time. The truck usage is to be given in terms of load factor (average capacity utilisation) and number of kilometres covered over the given period.
- The manager of the courier company can query the status of any particular consignment and the details of volume of consignments transported between any two branches and the corresponding revenue generated.
- The manager should be able to view the average waiting period for the consignments over a given period of time (day, month, or year) and that for various source destination pairs. This statistics is important for the manager, since he normally orders new trucks when the average waiting period for consignments becomes high due to non-availability of trucks. Also, the manager would like to see the average idle times of trucks over a given period time for future planning.
- The courier company would like the software to be modular and highly configurable, so that it can sell copies of the software to other courier companies in the country.

38. **Students' auditorium management software:** A college has a large (800 seating capacity) auditorium. The college has entrusted the management of the auditorium to the students' society. The students' society needs the following software to efficiently manage the various shows conducted in the auditorium and to keep track of the

accounts. The functionaries identified by the students' society to be responsible for the day-to-day operation of the software are the auditorium secretary and the president of the society.

Various types of social and cultural events are conducted in the auditorium. The auditorium secretary should have the overall authority of scheduling the shows, selecting and authorising the show managers, as well as the sales agents.

There are two categories of seats—balcony seats and ordinary seats. Normally balcony seats are more expensive in any show. The show manager fixes the prices of these two categories of seats for a specific show, depending on the popularity of a show. The show manager also determines the number of balcony and ordinary seats that can be put on sale. For each show, some seats are offered as complimentary gifts to important functionaries of the students' society and to VIPs which need to be entered into the system. The show manager also enters the show dates, the number of shows on any particular date and the show timings. It is expected that the software would support a functionality to let the show manager configure the different show parameters.

The auditorium secretary appoints a set of sales agents. The sales agents get a commission of 1 per cent of the total sales that they make for any show. The system should let the spectators query the availability of different classes of seats for a show on-line. For the convenience of the spectators, two ways of seat booking are supported. If a spectator pays ₹1000, a unique 10 digit id is generated and given to him. He can use this id to book seats for the shows on-line by using a web browser. For each seat booked for a show using the unique id, he would get a 10 per cent discount. A spectator can also book a seat for a single show only through regular payment. For on-line booking, a spectator would indicate for the type of the seat required by him, the requested seat should be booked if available, and the software should support printing out the ticket showing the seat numbers allocated. A spectator should be able to cancel his booking before 3 clear days of the show. In this case, the ticket price is refunded to him after deducting ₹5 as the booking charge per ticket. If a ticket is returned at least before 1 day of a show, a booking charge of ₹10 is deducted for ordinary tickets and ₹15 is deducted for balcony tickets. When a cancellation is made on the day of the show, there is a 50 per cent deduction.

The show manager can at any time query about the percentage of seats booked for various classes of seats and the amount collected for each class. When a sales person makes a sale, the computer should record the sales person's id in the sales transaction. This information would help in computing the total amount collected by each sales person and the commission payable to each sales person. These data can be queried by the show manager. Also, any one should be able to view the various shows that are planned for the next one month and the rates of various categories of seats for a show by using a web browser. The show manager should be able to view the total amount collected for his show as well as the sales agent-wise collection figures.

The accounts clerk should be able to enter the various types of expenditures incurred for a show including payment to artists and auditorium maintenance charges. The computer should prepare a balance sheet for every show and a comprehensive up-to-date balance sheet for every year. The different types of balance sheets should be accessible to the president of the student society only. Since the software product

should be as much low cost as possible, it is proposed that the software should run on a high-end PC and built using free system software such as Linux and Apache web server.

39. **Travel agency management software:** A travel agency requires to automate various book-keeping activities associated with its operations. The agency owns a fleet of vehicles and it rents these out to customers. Currently the company has the following fleet of vehicles:

- Ambassadors : 10 non-AC
- Tata Indica : 30 AC
- Tata Sumo : 5 AC
- Maruti Omni : 10 non-AC
- Maruti Esteem : 10 AC
- Mahindra Xylo : 10 AC

Only regular customers would be allowed to avail the on-line booking facility. To become a regular customer, a customer would need to deposit ₹5000 with the travel agency and also provide his address, phone number and few other details, which will be entered into the computer.

When a regular customer makes an on-line request for hiring a vehicle, he would be prompted to enter the date for which travel is required, the destination, and duration for which the vehicle is required. He should then be displayed the types of vehicles that are available, and the charges. For every type of vehicle, there is a per hour charge, and a per kilometre charge. These information would also be displayed. A car can be rented for a minimum of 4 hours. Once a customer chooses a vehicle, the number of the allotted vehicle and the driver's mobile numbers would be displayed.

After completion of travel, the customer would sign off the duty slip containing necessary information such as duration of hire and the kilometres covered and hand it over to the driver. This information would be entered into the computer system by the driver within 24 hours of completion of travel. The customer should be able to view the billing information as soon as the information in the duty slip have been entered into the computer. The amount chargeable to a customer is the maximum of per hour charge for the car times the number of hours used, plus the per kilometre charge times the number of kilometres run, subject to a minimum amount decided by the charge for 4 hours use of the car. An AC vehicle of a particular category is charged 25 per cent more than a non-AC vehicle of the same category. There is a charge of ₹500 for every night halt regardless of the type of the vehicle.

The travel agency acquires new vehicles at regular intervals. It is necessary to support a functionality using which the manager would be able to add a vehicle to the fleet of the vehicles as and when a new vehicle is procured. Old cars are condemned and sold off.

It should be possible to delete the car from the fleet and add the sales proceeds to the account. A car which is currently with the company can be in one of three states—it may have gone for repair, it may be available, and it may be rented out.

The manager should be able to view the following types of statistics—the price of the car, average amount of money spent on repairs for the car, average demand, revenue earned by renting out the car, and fuel consumption of the car. Based on these statistics, the company may take a decision about which vehicles are more profitable. The statistics can also be used to decide the rental charges for different types of vehicles.

40. **Students hall management center:** The IIT students' Hall Management Center (HMC) has requested us to develop the following software to automate various book-keeping activities associated with its day to day operations.
    - After a student takes admission, he/she would present a note from the admission unit to the clerk at HMC, along with his/her name, permanent address, contact telephone number, and a photograph. He/she is then allotted a hall, and also a specific room number. A letter indicating this allotted room should be issued to the concerned student.
    - Students incur mess charges every month. The mess manager would input to the software the total charges for each student in a month on mess account.
    - Each room has a fixed room rent. The rooms are either single-seated or twin-sharing. The newly constructed halls have higher rent compared to some of the older halls. Twin sharing rooms have lower rent.
    - Each hall provides certain amenities to the students such as reading rooms, play rooms, TV room, etc. A fixed amount is levied on each student on this count.
    - The total amount collected from each student of a hall towards mess charges is handed over to the mess manager every month. For this, the computer needs to print a sheet indicating the total amount due to each mess manager. Printed cheques are issued to each manager and signatures are obtained from each on the sheet.
    - Whenever a student comes to pay his dues, his total due should be computed as the sum of mess charge, amenity charge, and room rent and displayed. The amount would be paid by the student either in cash or cheque, and this would be entered by the accounts clerk into the system.
    - The students should be able to raise various types of complaints using a web browser in their room or in the Lab. The complaints can be repair requests such as fused lights, non-functional water taps, non-functional water filters, specific room repair, etc. They can also register complaints regarding the behaviour of attendants, mess staff, etc. For this, round-the-clock operation of the software is required and the down-time should be negligible. Considering that about 10,000 students live in hostels, the response time of the web site should be acceptable even under 1000 simultaneous clicks.
    - The HMC receives an annual grant from the institute for upkeep of the halls, gardens, and providing amenities to the students. The HMC chairman should be provided with a functionality that would support distribution of the grant among the different halls and cheques should be printed based on the grants made to the halls. The wardens of different halls should be able to enter their expenditure details against the allocations.
    - The controlling warden should be able to view the overall room occupancy.

- The warden of each hall should be able to find out the total occupancy of his hall and the number of vacant seats. He should also be able to view the complaints raised by the students and post his *action taken report* (ATR) to each complaint.
- The halls employ attendants and gardeners. These temporary employees receive a fixed pay on a per day basis. The Hall clerk enters any leave taken by an attendant or a gardener from at the terminal located at the hall office. At the end of every month a consolidated list of salary payable to each employee of the hall along with cheques for each employee is printed out.
- The HMC incurs petty expenses such as repair works carried out, newspaper and magazine subscriptions, etc. It should be possible to enter these expenses should be debited from its yearly grants.
- Whenever a new staff is recruited his details including his daily pay is entered. Whenever a staff leaves, it should be possible to delete his records. Upon a specific command from the controlling warden, the salaries for various employees of the HMC and halls should be computed and the salary slips and cheques should be printed for distribution to the employees.
- The warden should be able to view the statement of accounts any time. The warden would take a print out of the annual consolidated statement of accounts, sign and submit it to the Institute administration for approval and audit verification.

The software should be very secure to prevent the possibility of various types of frauds and financial irregularities.

41. Develop SA/SD diagrams for the following software required by a **video rental store.**

- The store has a large collection of video CDs and DVDs in VHS and MP4 format as well as music CDs.
- A person can become member by depositing ₹1000 and filling up name, address, and telephone number. A member can cancel his membership and take back his deposit, if he has no dues outstanding against him.
- Whenever the store purchases a new item, the details such as date of procurement and price are entered. The daily rental charge is also entered by the manager. After passage of a year, the daily rental charge is automatically halved.
- A member can take on loan at most one video CD and one music CD each time. The details are entered by a store clerk and a receipt indicating the daily rental charge is printed.
- Whenever a member returns his loaned item, the amount to be paid is displayed. After the amount is paid, the items are marked returned.
- If a customer loses or damages any item, the full price of the item is charged to him and the item is removed from the inventory.
- If an item lies unissued for more than a year, it is sold to the members at 10 per cent of the purchase price and the item is removed from the inventory.
- The manager can at any time check the profit/loss account.

42. Develop the SA/SD diagrams for the following **Elevator Controller** software.

The controller software of the elevator gets inputs from lift users through the push button switches mounted inside the elevator and near the lift door at each floor.

The controller generates output by giving commands to the motor controller and the lift door controller. At each floor, only two switches are installed. The switches are marked with up and down arrows indicating the request to go either in the up or down direction. Inside the lift, there is a panel of switches, with one switch labelled for each floor. Also there is an emergency stop switch. A user at a floor can request for the lift and indicate the required direction of travel by pressing the appropriate button. The requests for the lift arriving from various floors are queued by the controller and it serves the request in a shortest distance first manner, if the lift is idle. Once the lift stops at a floor, the user can press a switch labelled with the floor number to request the lift to go to that floor. After the floor request button is pressed, the lift controller times out after one minute and then starts to close the lift door. The successful closing of the lift door is indicated by the signals generated by a contact sensor. The lift starts to move in the required direction after the lift doors have completely closed. A user inside the lift can stop the lift doors from closing by pressing the emergency stop switch before the lift starts to move. Once the lift starts to move, pressing the emergency stop switch has no effect. At each floor, there is a touch sensor that indicates to the controller that a lift has reached the floor and the controller commands the motor to stop after a required floor is reached. After 30 seconds of reaching a floor, the lift door is opened by issuing a command to the door controller. If there is a power failure any time during a lift's movement, the lift reverts to a safe mode in which it shuts down the motor and a mechanical backup arrangement slides the lift to the ground floor and the manual door opening handle is enabled, which the user can use to open the lift doors.

43. Draw the level 0 (context level) and level 1 DFD representation of a simple alarm clock in which there is a button to advance alarm minute reading, a button to advance alarm hour reading, a button to set alarm, and a button to reset alarm.

44. Develop the SA/SD diagrams for the following **Hospital Management** software.

A large hospital needs a software to automate its various book-keeping activities. The hospital has a set of consultation rooms, a set of indoor rooms (called cabins) for resident patients. There are two types of doctors: regular and visiting consultants. The regular doctors sit in a consultation room for six hours every day. During 8 AM to 8 PM every day, at least three regular doctors are present. During 8 PM to 8 AM every day, at least one regular doctor is present. The software should provide facilities for the hospital administrator to input the names of the regular doctors employed by the hospital and their offered basic pay. The doctors also have a variable pay that is computed by multiplying the total number of patients they see over a month, multiplied by 100. The administrator should be able to generate pay slips for the doctors at the end of every month. The administrator should be able to delete the names of specific regular doctors who tender their resignation. The administrator also should be able to change the basic pay of a doctor. The administrator should be able to prepare weekly roster for the doctors. The roster contains the visiting hours for every doctor and the specific consultation room allotted to them for each day. The roster once prepared by the administrator is automatically uploaded at the hospital website and mails are sent to the doctors regarding their specific duty hours. The visiting consultants come for two hours slots twice a week and their roster is also

prepared by the administrator. The visiting consultants do not get a basic pay, but get ₹200 for every patient that they see. The patients can create their log-in account at the hospital website. The patients can see the specific doctors available for any day or over a week, and their qualifications and expertize. The patient can then proceed for booking an appointment with the doctor. The free slots available for the doctor would get displayed to him and he can book a slot of his convenience. But, for freezing the slot, he should pay a fee of ₹500 online. Once the patient visits the doctor at the appointed time, the doctor's diagnosis and prescriptions are stored online for future reference. In a similar fashion, the patients can book the indoor residence rooms (cabins) depending on availability. But, before they can freeze their booking, they would have to pay ₹3000 per day of booking online. There are also a set of nurses, who get fixed salaries. The names of the nurses and their salaries are entered by the administrator. The administrator should be able to delete the names of specific nurses or change their salaries. The duty allocation to the nurses for one week at a time is done by a head nurse. At any time, the administrator should be able to see the total number of patients who have visited the hospital for any given period, the total income, and the total salary outgo so far.

45. Develop the **Placement Assistant** software.

The main objective is to let students should be able to access details of available placements via Intranet. When there is a placement opportunity for which they wish to be considered, they would be able to apply for it electronically. This would cause a copy of their CV, which would also be held online to be sent to the potential employer. Details of interviews and placement offers would all be sent by e-mail. While some human intervention would be needed, the process need to be automated as far as possible. The following functionalities are to be supported:

- Enroll student
- Enroll company notify students
- Submit CV
- Notify job requirement
- Send matching CV
- Notify job offer
- Company feedback
- Student feedback