7 Design Engineering

7.1 Introduction

Design is a meaningful engineering representation of software. Design is the only way by which we can accurately **translate** the customer's **requirements** into a finished software **product** or system. In design engineering set of principles, concepts and practices are used in order to develop high quality software. Thus design serves as the basis for all the software engineering steps. Sometimes design is referred as a **core engineering activity** in software development life cycle. In this chapter we will get introduced with the systematic approach to design process. Then we will discuss how to bring quality in the software product. To implement right software product there should be some definite framework, and such a framework is defined in design concept. Finally we discuss the design model in which active participation of design elements in design procedure is discussed. We will begin our discussion with analysis and design model.

7.1.1 Analysis and Design model

- After analyzing and specifying all the requirements the process of software design begins. Each of the elements of analysis model is used to create the design model.
- The elements of analysis model are

1. Data Dictionary
2. Entity Relationship diagram
3. Data flow diagram
4. State transition diagram
5. Control specification
6. Process Specification

- The elements of design model are

  Data design

  Architectural Design

  Interface design

  Component-level design.

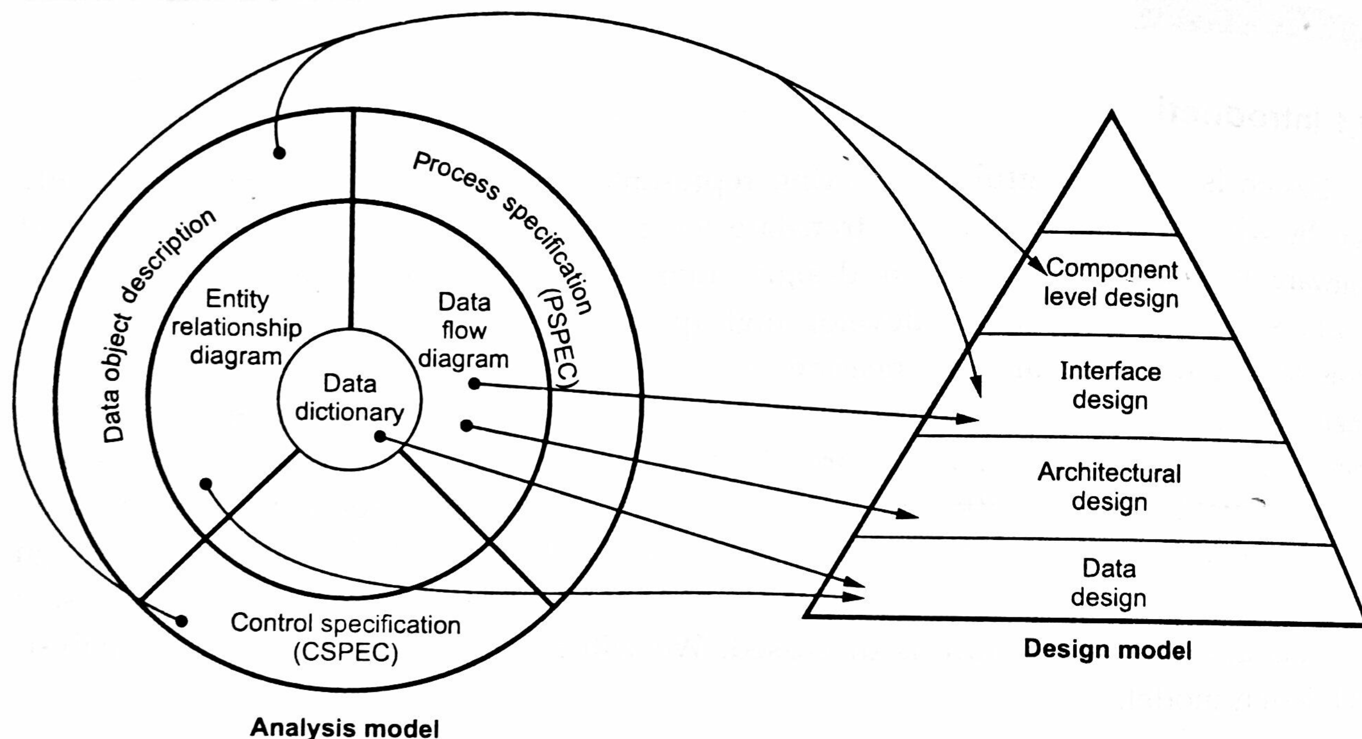


Fig. 7.1 Analysis and design model

- The *data design* is used to transform the information domain model of analysis phase into the data structures. These data structures play an important role in software implementation. The entity relationship diagram and data dictionary are used to create the data design model. In entity relationship diagram the relationships among the data objects is defined and in data dictionary detailed data contents are given, Hence ERD and data dictionary are used to implement the data design.
- The *architectural design* is used to represent the relationship between major structural elements with the help of some "design patterns." Hence data flow diagrams from analysis model serve as the basis for architectural design.
- The *'interface design'* describes how software interacts within itself. An interface means flow of information and specific type of behavior. Hence by using the data flow and control flow diagrams the interface design can be modeled.

- In the *'component-level design'* the structural elements of software architecture into procedural description of software components. Hence the 'component-level design' can be obtained using State Transition Diagrams (STD), Control Specification(CSPEC) and Process Specification (PSPEC).

7.2 Design Process and Design Quality

7.2.1 Design Process

- Design Process is a sequence of steps carried through which the user requirements are translated into a system or software model.
- The design is represented at high level of abstraction.

Following figure shows the design activities -

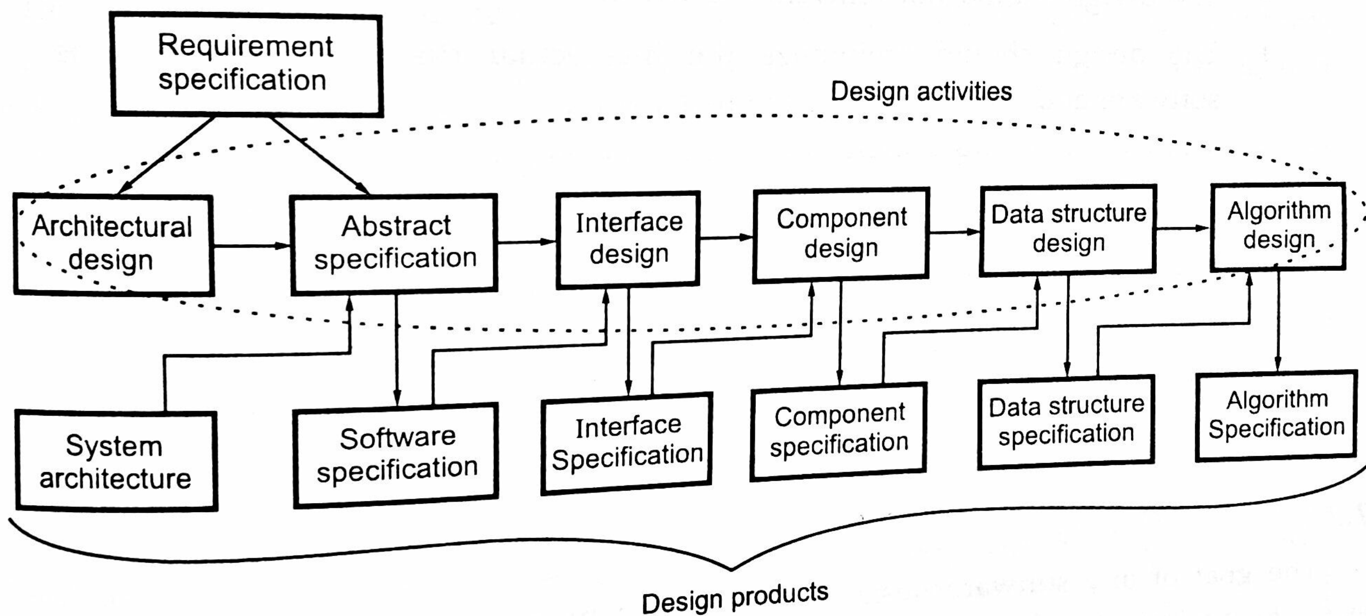


Fig. 7.2 Design Process

1. From requirements specification the architectural design and abstract specification is created. In architectural design the subsystem components can be identified. And the abstract specification is used to specify the subsystems.
2. Then the interfaces between the subsystems are designed which is called interface design.
4. In component design of subsystems components is done.
5. Once the components of the system are identified and designed the decision on use of particular data structures is taken. And the data structure is designed to hold the data.

6. For performing the required functionality, the appropriate algorithm is designed. These algorithms should suit the data structures selected for software product.

The design process occurs in iterations so that subsequent refinement is possible. At each design step the corresponding specifications are created. Throughout the design process the quality of evolving design is assessed with the help of formal technical reviews and design walkthroughs.

7.2.2 Design Principles

Davis suggested a set of principles for software design as:

- The design process should not suffer from "tunnel vision".
- The design should be traceable to the analysis model.
- The design should not reinvent the wheel.
- The design should "minimize the intellectual distance" between the software and the problem in the real world.
- The design should exhibit uniformity and integration.
- The design should be structured to accommodate change.
- The design should be structured to degrade gently.
- Design is not coding and coding is not design.
- The design should be assessed for quality.
- The design should be reviewed to minimize conceptual errors.

7.2.3 Design Quality

The goal of any software design is to produce high quality software. In order to evaluate quality of software there should be some predefined rules or criteria that need to be used to assess the software product. Such criteria serve as characteristics for good design. The quality guidelines are as follows –

1. The design architecture should be created using following issues –
   - The design should be created using **architectural styles and patterns.**
   - Each component of design should posses **good design characteristics**
   - The implementation of design should be **evolutionary,** so that testing can be performed at each phase of implementation.
2. In the design the data, architecture, interfaces and components should be clearly represented.
3. The design should be **modular**. That means the subsystems in the design should be logically partitioned.

4. The **data structure** should be appropriately chosen for the design of specific problem.
5. The **components** should be used in the design so that functional independency can be achieved in the design.
6. Using the information obtained in software **requirement analysis** the design should be created.
7. The **interfaces** in the design should be such that the complexity between the connected components of the system gets reduced. Similarly interface of the system with external interface should be simplified one.
8. Every design of the software system should convey its **meaning** appropriately and effectively.

7.2.4 Design Quality Attributes

The design quality attributes popularly known as **FURPS** (**F**unctionality, **U**sability, **R**eliability, **P**erformance and **S**upportability) is a set of criteria developed by Hewlett and Packard. Following table represents meaning of each quality attribute

Quality Attribute	Meaning
Functionality	Functionality can be checked by assessing the set of **features and capabilities** of the functions. The functions should be **general** and should not work only for particular set of inputs. Similarly the **security** aspect should be considered while designing the function.
Usability	The usability can be assessed by knowing the **usefulness** of the system.
Reliability	Reliability is a measure of frequency and severity of **failure**. Repeatability refers to the consistency and repeatability of the measures. The mean time to failure (**MTTF**) is a metric that is widely used to measure the product's performance and reliability.
Performance	It is a measure that represents the response of the system. Measuring the performance means measuring the processing speed, memory usage, response time and efficiency.
Supportability	It is also called maintainability. It is the ability to adopt the enhancement or changes made in the software. It also means the ability to withstand in a given environment.

7.3 Design Concepts

The software design concept provides a framework for implementing the right software.

Following issues are considered while designing the software –

1. Abstraction –

The abstraction means an ability to cope with the complexity. At each stage of software design process levels of abstractions should be applied to refine the software solution. At the higher level of abstraction, the solution should be stated in broad terms and in the lower level more detailed description of the solution is given.

While moving through different levels of abstraction the procedural abstraction and data abstraction are created.

The procedural abstraction gives the named sequence of instructions in the specific function. That means the functionality of procedure is mentioned by its implementation details are hidden. For example: *Search the Record* is a procedural abstraction in which implementation details are hidden(i.e. Enter the name, compare each name of the record against the entered one, if a match is found then declare success!! Otherwise declare 'name not found')

In data abstraction the collection of data objects is represented. For example for the procedure *search* the data abstraction will be *Record*.The record consists of various attributes such as Record ID, name, address and designation.

2. Modularity –

- The software is divided into separately named and addressable components that called as modules.
- Meyer defines five criteria that enable us to evaluate a design method with respect to its ability to define an effective modular system:

  Modular decomposability : A design method provides a systematic mechanism for decomposing the problem into sub-problems. This reduces the complexity of the problem and the modularity can be achieved.

  Modular composability : A design method enables existing design components to be assembled into a new system.

  Modular understandability: A module can be understood as a stand alone unit. It will be easier to build and easier to change.

  Modular continuity: Small changes to the system requirements result in changes to individual modules, rather than system-wide changes.

Modular protection: An aberrant condition occurs within a module and its effects are constrained within the module.

3. Architecture –

Architecture means representation of **overall structure** of an integrated system. In architecture various components interact and the data of the structure is used by various components. These components are called **system elements**. Architecture provides the basic framework for the software system so that important framework activities can be conducted in systematic manner.

In architectural design various system models can be used and these are

Model	Functioning
Structural model	Overall architecture of the system can be represented using this model
Framework model	This model shows the architectural framework and corresponding applicability.
Dynamic model	This model shows the reflection of changes on the system due to external events.
Process model	The sequence of processes and their functioning is represented in this model
Functional model	The functional hierarchy occurring in the system is represented by this model.

4. Refinement –

- Refinement is actually a process of elaboration.
- Stepwise refinement is a top-down design strategy proposed by Niklaus WIRTH.
- The architecture of a program is developed by successively refining levels of procedural detail.
- The process of program refinement is analogous to the process of refinement and partitioning that is used during requirements analysis.
- Abstraction and refinement are complementary concepts. The major difference is that - in the abstraction low-level details are suppressed. Refinement helps the designer to elaborate low-level details.

5. Pattern –

According to **Brad Appleton** the design pattern can be defined as – It is a named nugget(something valuable) of insight which conveys the essence of proven solution to a recurring problem within a certain context.

In other words, design pattern acts as a design solution for a particular problem occurring in specific domain. Using design pattern designer can determine whether-

- Pattern can be **reusable**
- Pattern can be used for **current work**
- Pattern can be used to **solve similar kind of problem** with different functionality.

6. Information hiding –

It is the characteristics of module in which major design decisions can be hidden from all others. Such hiding is necessary because information of one module cannot be accessed by another module. The advantage of information hiding is that modifications during testing and maintenance can be made independently without affecting the functionality of other modules.

7. Functional independence –

Functional independence can be achieved by modularity, abstraction and information hiding. Functional independence obtained by creating each module that is performing only one specific task. And such a module should be interconnected to other components by a simple interface. Effective modularity helps in achieving the functional independence. In short, Functional independence is a key to good design and good design leads a quality in software product. The functional independence is assessed using two factors cohesion and coupling.

Cohesion

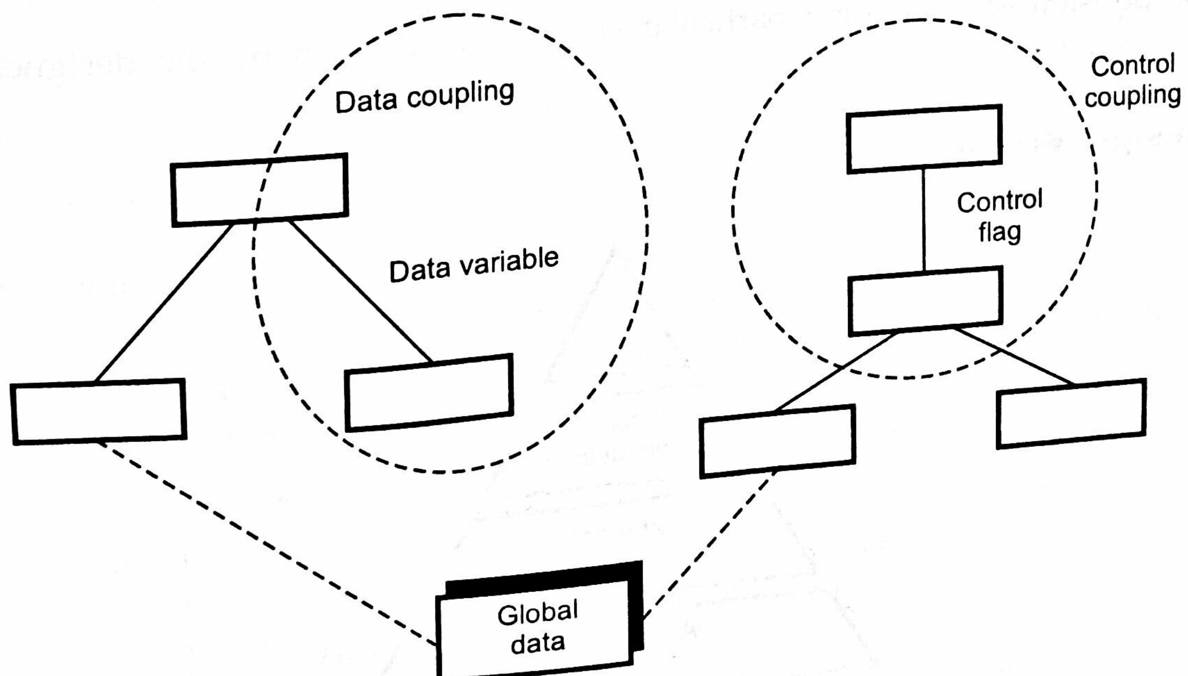
- With the help of cohesion the information hiding can be done.
- A cohesive module performs only "one task" in software procedure with little interaction with other modules. In other words cohesive module performs only one thing.
- Different types of cohesion are :
  1. **Coincidentally cohesive** – The modules in which the set of tasks are related with each other loosely then such modules are called coincidentally cohesive.
  2. **Logically cohesive** – A module that performs the tasks that are logically related with each other is called logically cohesive.

3. **Temporal cohesion** – The module in which the tasks need to be executed in some specific time span is called temporal cohesive.
4. **Procedural cohesion** – When processing elements of a module are related with one another and must be executed in some specific order then such module is called procedural cohesive.
5. **Communicational cohesion** – When the processing elements of a module share the data then such module is communicational cohesive.

- The goal is to achieve high cohesion for modules in the system.

Coupling

- Coupling effectively represents how the modules can be "connected" with other module or with the outside world.
- Coupling is a measure of interconnection among modules in a program structure.
- Coupling depends on the interface complexity between modules.
- The goal is to strive for lowest possible coupling among modules in software design.
- The property of good coupling is that it should reduce or avoid change impact and ripple effects. It should also reduce the cost in program changes, testing, and maintenance.



- Various types of coupling are :
  i) **Data coupling** – The data coupling is possible by parameter passing or data interaction.
  ii) **Control coupling** – The modules share related control data in control coupling.
  iii) **Common coupling** – In common coupling common data or a global data is shared among the modules.
  iv) **Content coupling** – Content coupling occurs when one module makes use of data or control information maintained in another module.

8. Refactoring –

Refactoring is necessary for simplifying the design without changing the function or behaviour. Fowler has defined refactoring as *the process of changing a software system in such a way that the external behaviour of the design do not get changed, however the internal structure gets improved.*

Benefits of refactoring are –

- The **redundancy** can be achieved.
- **Inefficient algorithms** can be eliminated or can be replaced by efficient one.
- Poorly constructed or **inaccurate data structures** can be removed or replaced.
- Other **design failures** can be rectified.

The decision of refactoring particular component is taken by the designer of the software system.

7.4 Design Models

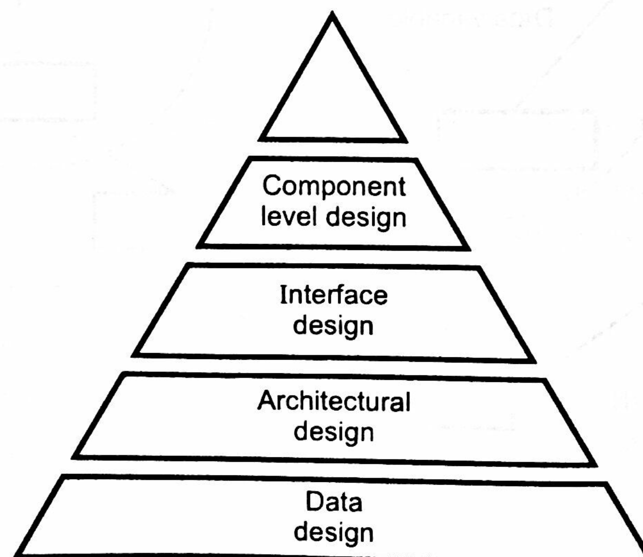


Fig. 7.4 Design model

- The design model is represented as pyramid. The pyramid is a stable object. Representing design model in this way means that the software design should be stable.
- The design model has broad foundation of data design, stable mid-region with architectural and interface design and the sharp point to for component level design.
- The design model represents that the software which we create should be stable such that any changes should not make it collapsed. And from such a stable design a high quality software should be generated.

7.4.1 Data Design Element

The data design represents the **high level of abstraction**. This data represented at data design level is **refined gradually** for implementing the computer based system. The data has great impact on the architecture of software systems. Hence structure of data is very important factor in software design. Data appears in the form of **data structures and algorithms** at the program **component level**. At the **application level** it appears as the **database** and at the **business level** it appears as **data warehouse and data mining**. Thus data plays an important role in software design.

7.4.2 Architectural Design Element

The architectural design gives the layout for overall view of the software. Architectural model can be built using following sources –

- Data flow models or class diagrams
- Information obtained from application domain
- Architectural patterns and styles.

7.4.3 Interface Design Elements

Interface Design represents the **detailed design** of the software system. In interface design how **information flows** from one component to other component of the system is depicted. Typically there are three types of interfaces-

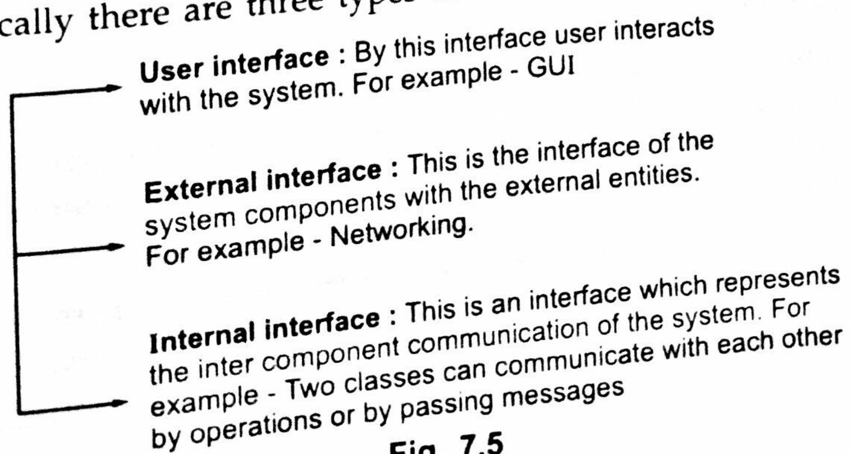


Fig. 7.5

7.4.4 Component Level Design Elements

The component level design is more detailed design of the software system along with the specifications. The component level design elements describe the internal details of the component. In component level design all the local data objects, required data structures and algorithmic details and procedural details are exposed.

7.5 Design Document

The design document can be created as follows

Design Specification

1.0 Overall scope

1.1 Data design

1.2 Architectural design

1.3 External and internal interfaces

1.4 Requirements cross reference

1.5 Design constraints

1.6 Supplementary data

1.7 Appendix

1.8 Installation manuals

- The design document is used to represent various aspects of design model.
- In this document first of all overall scope of the design effort is described. The information presented here is used from the SRS.
- Then in data design database structure, any external file structure, internal data structure, cross reference of data objects to files is defined.
- The architectural design shows how analysis model builds the program architecture. Sometimes structure charts are used to represent the module hierarchy.
- Then internal and external program interfaces are given. In some cases a detailed prototype of a GUI may be represented.
- The requirement cross reference is given in order to ensure that all requirements are satisfied by the software design. The cross references also indicate which component are critical for implementation. The test documentation is also included in the design document.
- Under design constraints the information such as memory requirements, special requirement for assembling or packaging the software, requirement of virtual memory, high speed requirement is given.

8 Architectural Design

8.1 Introduction

As we have seen in previous chapter, software design is a process in which user requirements are transformed into a system model. Such transformation can be carried out in various phases such as data design, architectural design, interface design and component design. Out of which architectural design is the focus of this chapter. **Architectural design** is a design created to represent the data and program components that are required to build the computer based systems. Architectural design is a specialized activity in which using specific architectural style and by considering the system's structure and properties of system components a large and complex system is built. The person who is responsible to design such system is called **software architect** in software engineering. The architectural design gives a layout of the system that is to be built. In short, the program structure is created during architectural design along with the description of component properties and their inter-relationship.

In this chapter first of all we will understand the concept of software architecture. Then we will discuss the role of data in the architectural design by means of data design. We will also discuss very interesting architectural styles and pattern that are selected for specific architectural built. Finally we will understand the complete process of architectural design.

8.2 Software Architecture

The architectural design is the design process for identifying the subsystems making up the system and framework for subsystem control and communication.

The goal of architectural design is to establish the overall structure of software system. Architectural design represents the link between design specification and actual design process.

> *Software Architecture is a structure of systems which consists of various components, externally visible **properties** of these components and the **inter-relationship** among these components*

Importance of Software Architecture

There are three reasons why the software architecture is so important?

1. Software architecture gives the representation of the computer based system that is to be built. Using this system model even the **stakeholders** can take **active part** in the software development process. This helps in clear specification/understanding of requirements.
2. Some **early design decisions** can be taken using software architecture and hence system performance and operations remain under control.
3. The software architecture gives a clear cut **idea** about the computer based system which is to be built.

8.2.1 Structural Partitioning

The program structure can be partitioned horizontally or vertically.

Horizontal partitioning

Horizontal partitioning defines separate branches of the modular hierarchy for each major program function.

Horizontal partitioning can be done by partitioning system into : input, data transformation (processing) and output.

In horizontal partitioning the design making modules are at the top of the architecture.

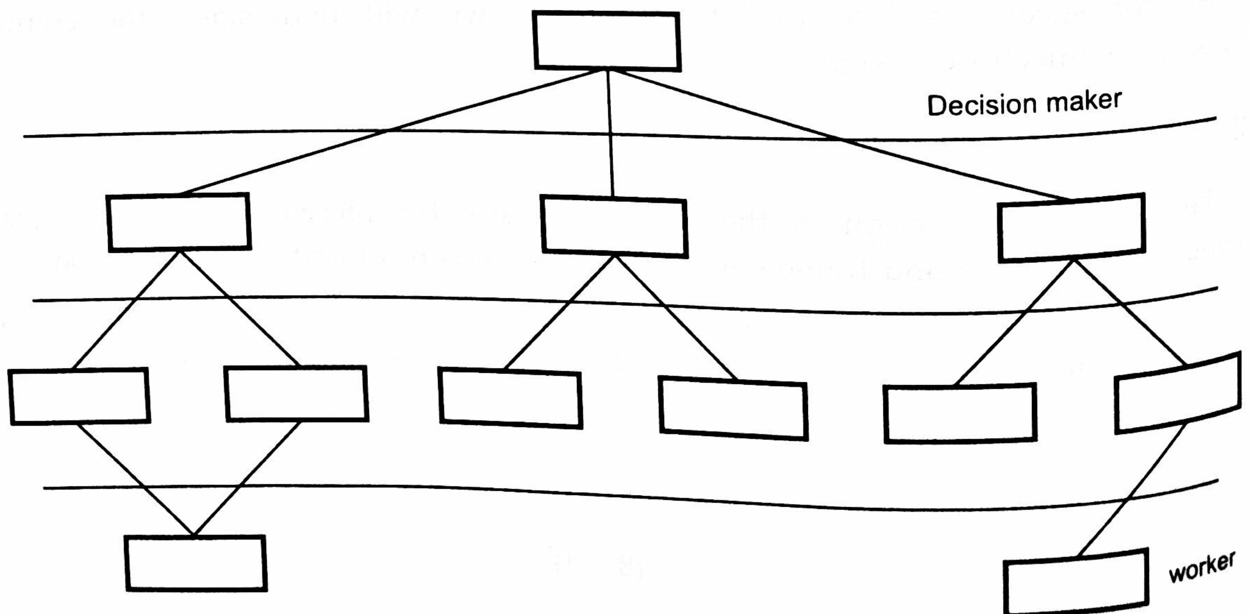


Fig. 8.1 Horizontal partitioning

Advantages of horizontal partition

1. These are easy to test, maintain and extend.
2. They have fewer side effects in change propagation or error propagation.

Disadvantage of horizontal partition

More data has to be passed across module interfaces which complicate the overall control of program flow.

Vertical partitioning

Vertical partitioning suggests the control and work should be distributed top-down in program structure.

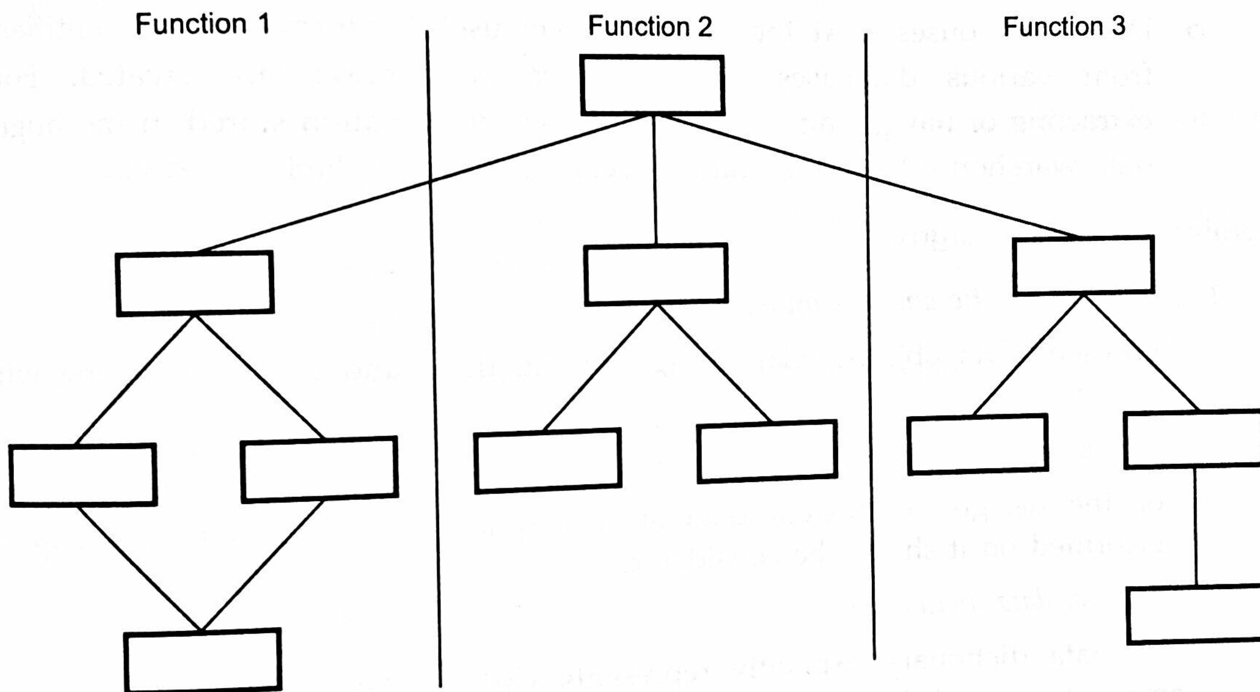


Fig. 8.2 Vertical partitioning

In vertical partitioning

- Define separate branches of the module hierarchy for each major function.
- Use control modules to co-ordinate communication between functions.

Advantages of vertical partition

1. These are easy to maintain the changes.
2. They reduce the change impact and error propagation.

8.3 Data Design

- Data design is basically the model of data that is represented at the high level of abstraction.
- The data design is then progressively refined to create implementation specific representations.
- Various elements of data design are
  - o Data object – The data objects are identified and relationship among various data objects can be represented using entity relationship diagrams or data dictionaries.
  - o Databases – Using software design model, the data models are translated into data structures and databases at the application level.
  - o Data warehouses – At the business level useful information is identified from various databases and the data warehouses are created. For extracting or navigating the useful business information stored in the huge data warehouse then data mining techniques are applied.

Guideline for data design✓

1. *Apply systematic analysis on data*

   Represent data objects, relationships among them and data flow along with the contents.

2. *Identify data structures and related operations*

   For the design of efficient data structures all the operations that will be performed on it should be considered.

3. *Establish data dictionary*

   The data dictionary explicitly represents various data objects, relationships among them and the constraints on the elements of data structures.

4. *Defer the low-level design decisions until late in the design process*

   Major structural attributes are designed first to establish an architecture of data. And then low-level design attributes are established.

5. *Use information hiding in the design of data structures*

   The use of information hiding helps in improving quality of software design. It also helps in separating the logical and physical views.

6. *Apply a library of useful data structures and operations*

   The data structures can be designed for reusability. A use of library of data structure templates (called as abstract data types) reduces the specification and design efforts for data.

7. *Use a software design and programming language to support data specification and abstraction*

The implementation of data structures can be done by effective software design and by choosing suitable programming language.

8.4 Architectural Styles and Pattern

8.4.1 Architectural Styles

- The **architectural model or style** is a pattern for creating the system architecture for given problem. However, most of the large systems are heterogeneous and do not follow single architectural style.
- System categories define the architectural style

  1. Components : They perform a function.

  For example: Database, simple computational modules, clients, servers and filters.

  2. Connectors : Enable communications. They define how the components communicate, co-ordinate and co-operate.

  For example Call, event broadcasting, pipes

  3. Constraints : Define how the system can be integrated.

  4. Semantic models : Specify how to determine a system's overall properties from the properties of its parts.

- The commonly used architectural styles are

  1. Data centered architectures
  2. Data flow architectures
  3. Call and return architectures
  4. Object oriented architectures
  5. Layered architectures

8.4.1.1 Data Centered Architectures

In this architecture the data store lies at the centre of the architecture and other components frequently access it by performing add, delete and modify operations. The client software requests for the data to central repository. Sometime the client software accesses the data from the central repository without any change in data or without any change in actions of software actions.

Data centered architecture posses the property of interchangeability. Interchangeability means any component from the architecture can be replaced by a new component without affecting the working of other components.

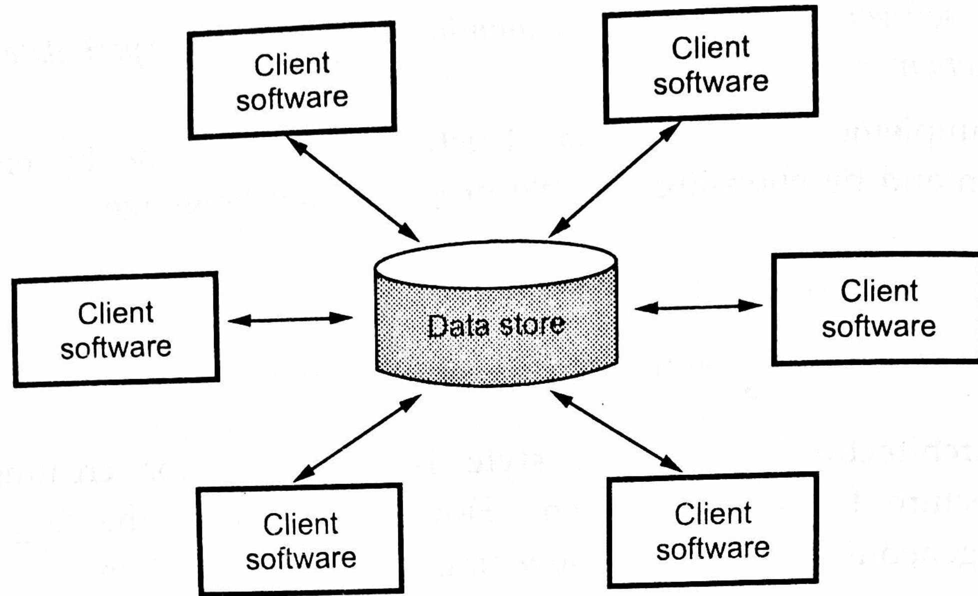


Fig. 8.3 Data centered architecture

In data centered architecture the data can be passed among the components.

In data centered architecture

Components are : Database elements such as tables, queries.

Communication are : By relationships

Constraints are : Client software has to request central data store for information.

8.4.1.2 Data Flow Architectures

In this architecture series of transformations are applied to produce the output data. The set of components called filters are connected by pipes to transform the data from one component to another. These filters work independently without a bothering about the working of neighbouring filter.

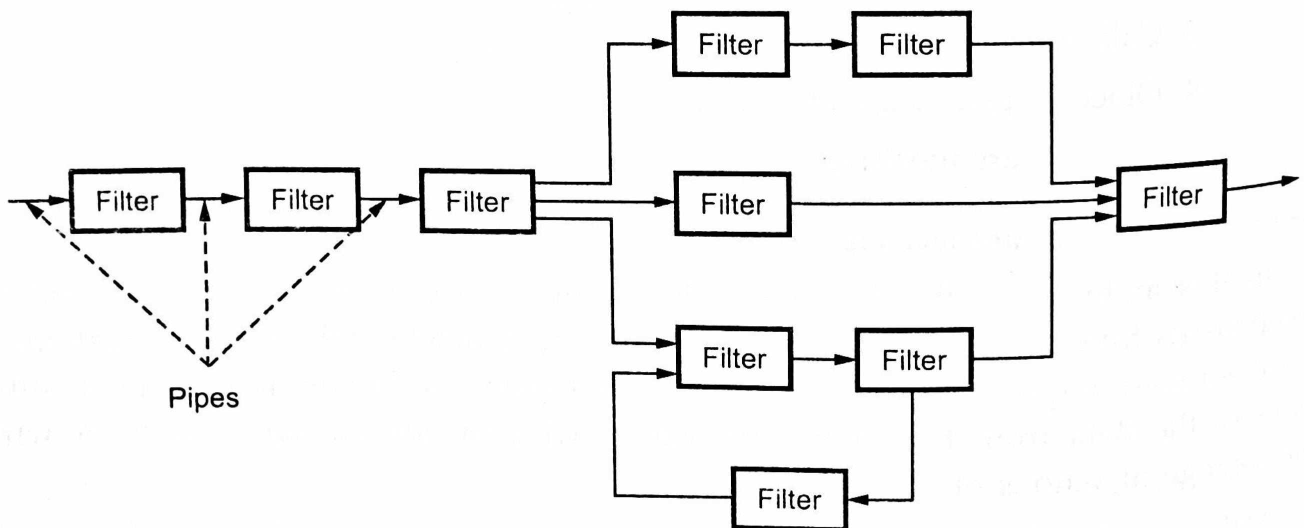


Fig. 8.4 Pipes and Filters

If the data flow degenerates into a single line of transforms, it is termed as batch sequential.

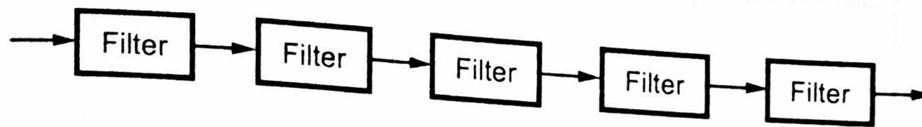


Fig. 8.5 Batch sequential

In this pattern the transformation is applied on the batch of data.

8.4.1.3 Call and Return Architecture

The program structure can be easily modified or scaled. The program structure is organized into modules within the program. In this architecture how modules call each other. The program structure decomposes the function into control hierarchy where a main program invokes number of program components.

In this architecture the hierarchical control for call and return is represented.

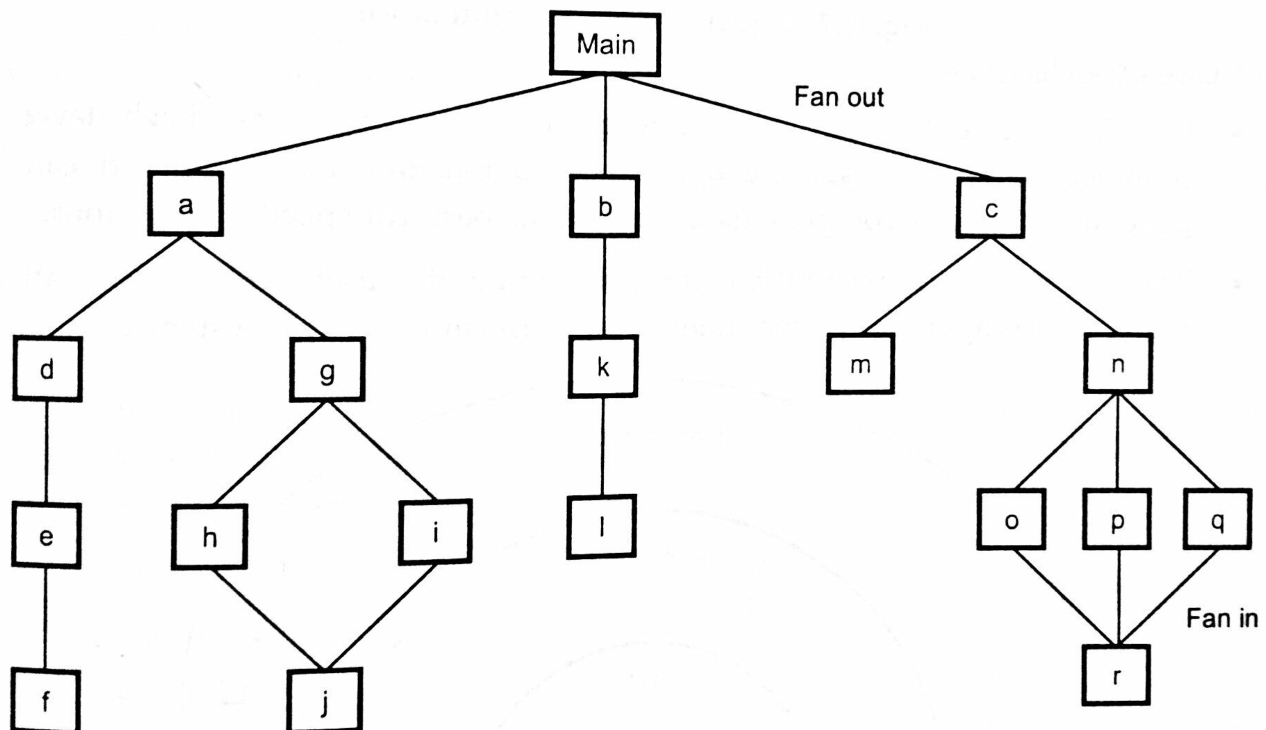


Fig. 8.6 Call and return architecture

8.4.1.4 Object Oriented Architecture

In this architecture the system is decomposed into number of interacting objects.

These objects encapsulate data and the corresponding operations that must be applied to manipulate the data.

The object oriented decomposition is concerned with identifying objects classes, their attributes and the corresponding operations. There is some control models used to co-ordinate the object operations.

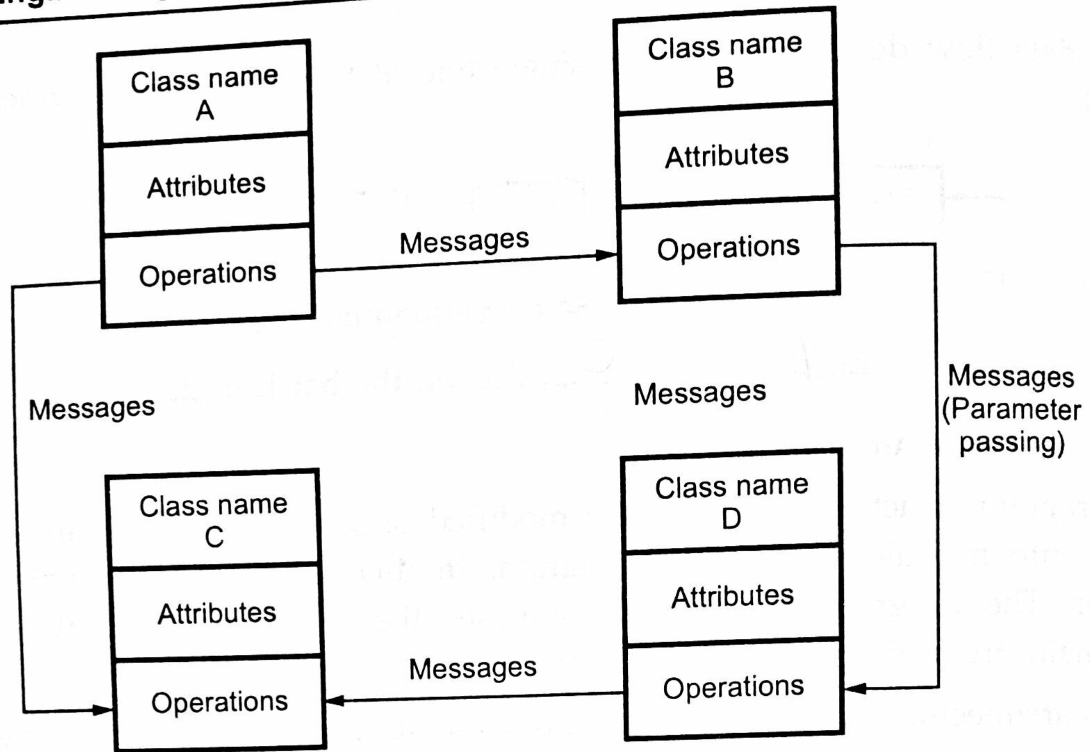


Fig. 8.7 Object oriented architecture

8.4.1.5 Layered Architecture

- The layered architecture is composed of different layers. Each layer is intended to perform specific operations so machine instruction set can be generated. Various components in each layer perform specific operations.
- The outer layer is responsible for performing the user interface operations while the components in the inner layer perform operating system interfaces.

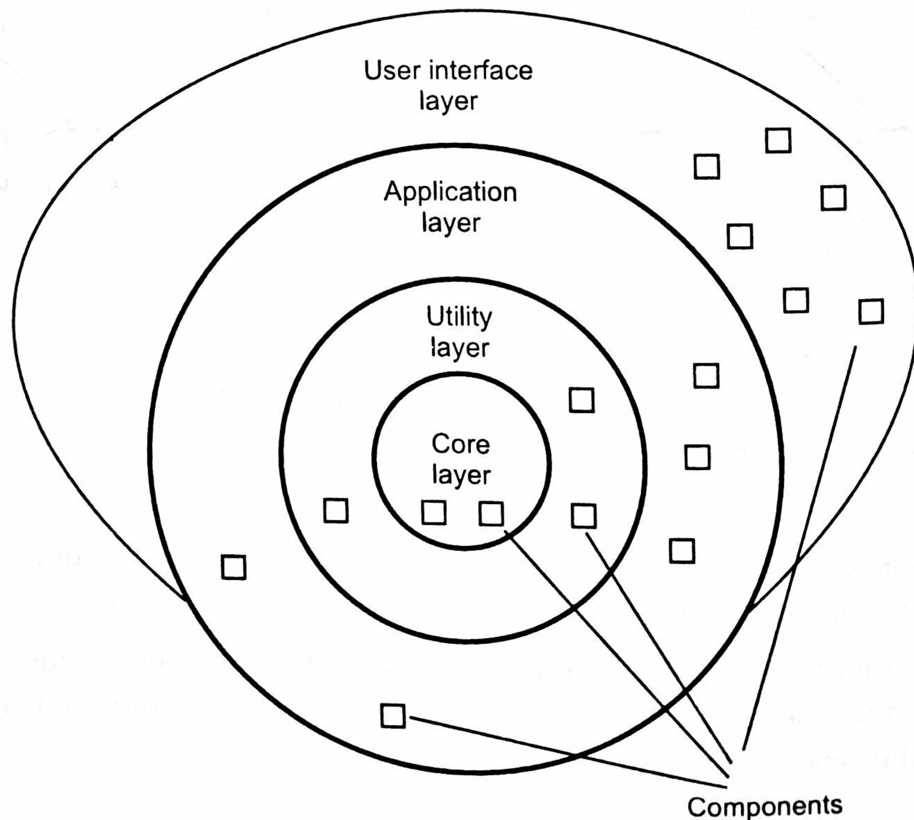


Fig. 8.8 Layered architecture components

- The components in intermediate layer perform utility services and application software functions.

8.4.2 Architectural Patterns

In this section we will understand *what are architectural patterns ?* The architectural pattern is basically an approach for handling behavioural characteristics of software systems. Following are the architectural pattern domains

1. Concurrency

Concurrency means handling multiple tasks in parallel. For example in operating system, **multiple tasks** are executed in parallel. Hence concurrency is a pattern which represents that the system components can interact with each other in parallel. The benefit of this pattern is that system **efficiency** can be achieved.

2. Persistence

Continuity in the data can be maintained by the persistence pattern. In other words the data used in earlier execution can be made available further by storing it in files or in **databases**. These files/databases can be modified in the software system as per the need. In **object oriented** system the values of all attributes various operations that are to be executed are persistent for further use. Thus broadly there are two patterns. i) Database management pattern ii) Application level pattern.

3. Distribution

Distribution pattern refers to the way in which the system components communicate with each other in distributed systems. There are two major problems that occur in distribution pattern

- The nature of interconnection of the components
- The nature of communication

These problems can be solved by other pattern called **broker pattern.** The broker pattern lies between server and client components so that the client server communication can be established properly. When client want some service from server, it first sends message to broker. The broker then conveys this message to server and completes the connection. Typical example is CORBA. The CORBA is a distributed architecture in which broker pattern is used.

8.5 Architectural Design

In architectural design at the initial stage a **context model** is prepared. This model defines the external entities that interact with the software. Along with this model the nature of software interaction with external entities is also described. The context

model is prepared by using information obtained from analysis model and requirement specification. After that the designer creates **structure of the system** by defining and refining software components. Thus process of creations of context model and structural model of the system is iteratively carried out until and unless complete architectural model of the system gets created. Let us discuss how an architectural design gets generated using some simple representations.

8.5.1 Representing System in Context

We have already discussed in chapter 6, how to create a context model for the given software system. As per our discussion, context model is a graphical model in which the environment of the system is defined by showing the external entities that interact with the software system.

In architectural design the Architectural Context Diagram(ACD) is created. The difference between context model and architectural context diagram is that in ACD the nature of interaction is clearly described. Following are the basic terminologies associated with architectural context diagram.

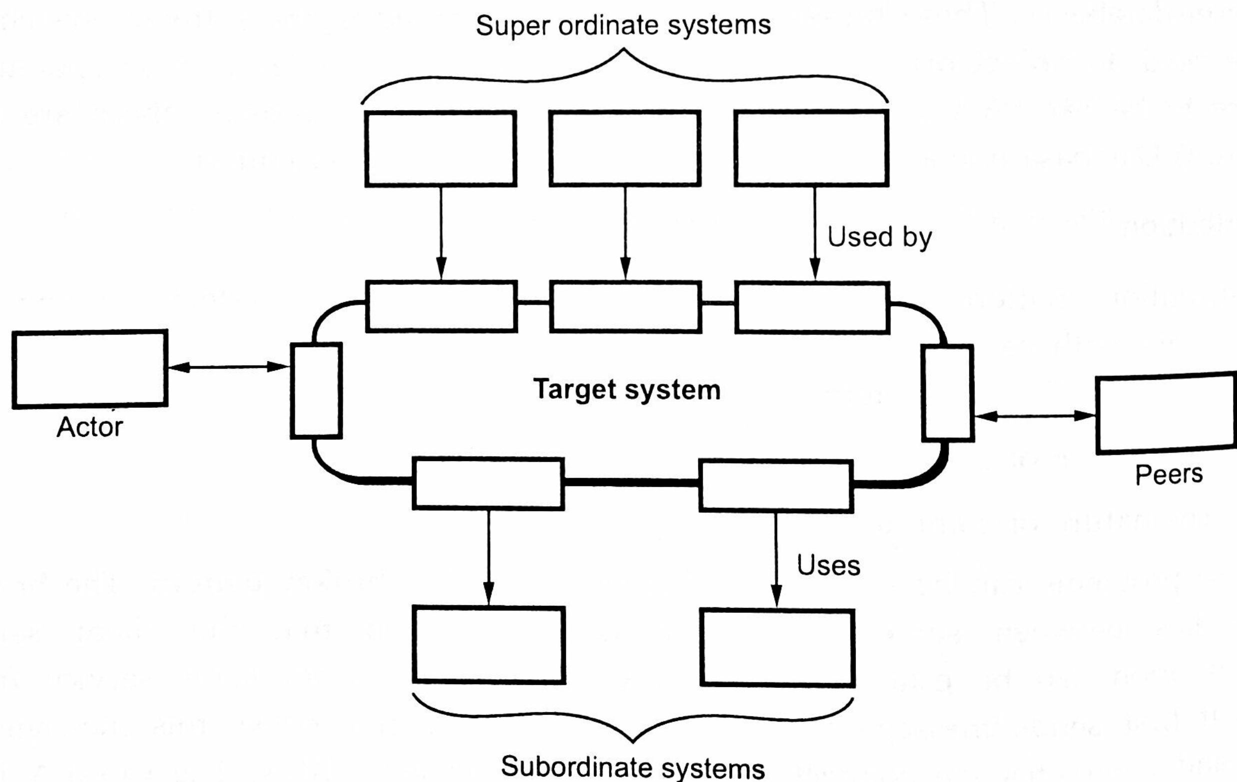


Fig. 8.9 Architectural context diagram

Target system : The target system is a computer based system for which the architectural context diagram has to be prepared.

Super ordinate systems : These systems are created at higher level of processing when the target system is being developed.

Sub ordinate systems : These systems are used by the target system for processing of the data that are necessary to complete target system functionality

Actors : These are the systems or entities that interact with the target system for producing or consuming information of the target system.

Peer-level systems: These are the systems that interact on peer to peer basis.

For example

Consider the *Inventory Control System* for which the Architectural Context Diagram can be prepared as below

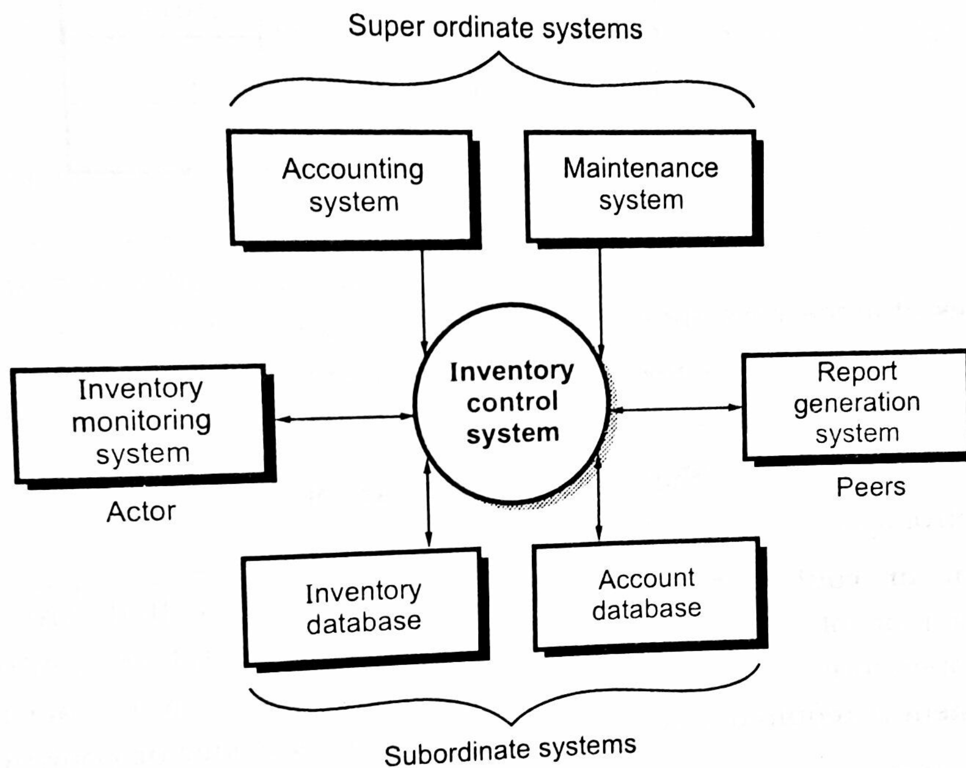


Fig. 8.10 ACD for inventory control system

8.5.2 Defining Archetypes

Defining archetype is a **basic step** in architectural design, more precisely in functionality based design. Archetype is a **core abstraction** using which the system can be structured. Using archetypes, a **small set** of entities that describe the major part of system behaviour can be described. Typically archetypes are the stable elements and they do not change even though system undergo through various changes. Identifying archetypes is a critical task and it requires well experienced architect. Following figure represents various archetypes.

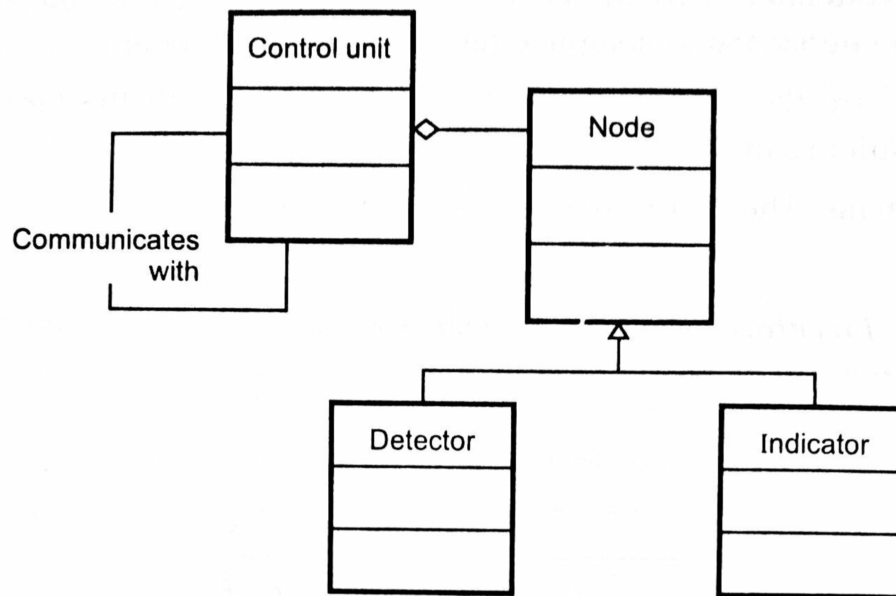


Fig. 8.11 Archetypes

Various types of archetypes are

Point or node : It refers to the highest level of abstraction in which there is a cohesive collection of input and output functionalities.

Detector : Detectors capture the core functionalities of the system. For example, in temperature control system sensors for temperature is a sensor.

Control unit or controller : Controllers are the entities that are useful for controlling behaviour of the system. For example, in temperature control system. When the temperature exceeds beyond some threshold value alarming and dis-alarming system is required. Such a system acts as a controller or control unit.

Indicator or output : It represents the generic output functionalities. For example, monitoring system of any computer based system acts as an indicator.

In software engineering archetype is a number of **major components** that are used to describe the system which we want to build.

8.5.3 Refining Architecture into Components

To create full structure of the system it is required to refine the software architecture into components. Hence it is necessary to identify the components of the system. The components can be identified from **application domain** or from **infrastructure domain**. The architectural designer has to identify these components from these domains. There are two methods by which the components can be identified.

1. The **data flow diagram** is drawn from which the specialized components can be identified. Such components are the components that process the data flow across the interfaces.
2. The components can be the entities that follow following **functionalities** –

   External communication : The components that take part in the communication with external entities are the communication components.

   Control panel processing : These components perform all control panel management activities.

   Detection : These are the components that perform detection activities.

   Indicator management : These are the components that perform the output controlling activities.

8.5.4 Defining Instantiations of the System

In order to model a structure of the system simply refining the software into components is not sufficient. Further refinement is necessary by instantiation of the system. Instantiation of the system begins with identification of major components and then identification of its functionalities, characteristics and constraints is carried out in order to refine the system to greater extent.

Finally with sufficient detailing, the architectural model of the system gets ready.

8.6 Transform and Transaction Mapping

8.6.1 Transform Mapping

The transform mapping is a set of design steps applied on the DFD in order to map the transformed flow characteristics into specific architectural style.

Design steps for transform mapping

We will consider an example of security home system and apply the design steps for performing transform mapping.

Example : Home security system

Security system software is prepared for the homeowner for home security purpose. After installation of this system it needs to be configured. This system has control panel through which the home owner can interact with it using keypad and functional keys. The sensors are connected to the system and to monitor the status of sensors.

While installing the security system software the control panel is used to program and configure the system. During configuration each sensor is assigned a number and

type, a master password is programmed for alarming and de-alarming the system. The telephone numbers of emergency services are programmed in the system as input for dialing when a sensor event occurs.

On occurrence of sensor event, it is recognized first and then an alarm which is attached to the system starts ringing. After a delay time (which is specified by the homeowner during the configuration) the software dials the telephone number, provides the information about the nature of event and its location.

The telephone number will be redialed every 20 seconds until telephone connection is obtained.

Step 1 : Review the fundamental system model to identify the information flow

The fundamental system model can be represented by level 0 DFD and supporting information. This supporting information can be obtained from the two important documents called 'system specification' and 'software requirement specifications'. Both of them describe the information flow and structure at software interface.

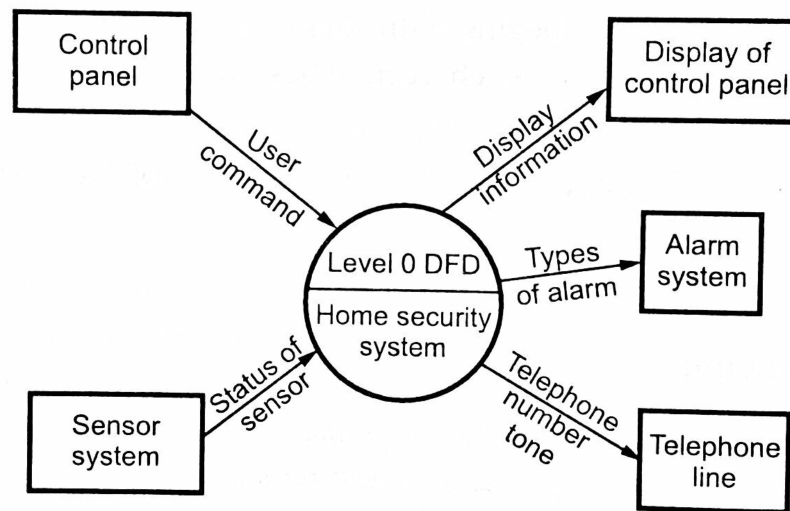


Fig. 8.12 Level 0 DFD

Step 2 : Review and refine the data flow diagrams for the software –

The data flow diagrams are analyzed and refined into next higher levels. Each transform in the data flow diagrams impose relatively high level cohesion. That means after applying the certain transformation the process in the DFD performs a single distinct function.

For example, the DFD is refined to level 1 to the working of the system. Further the level 2 DFD is drawn in which the detailing of sensor monitoring system is done.

Control panel
User command
Configuration system
Invoke to configure systems
Configuration data
Configuration information store
Interact with user
Configuration data
Master password
On/off
Activate or deactivate system
Activate or deactivate message
Display status with messages
Display info.
Display of control panel
Validate password
User validity message
Configuration data
Information about sensor
Sensor system
Status of sensor
Sensor monitoring system
Type of alarm
Alarm system
Telephone number tone
Telephone line

Fig. 8.13 Level 1 DFD

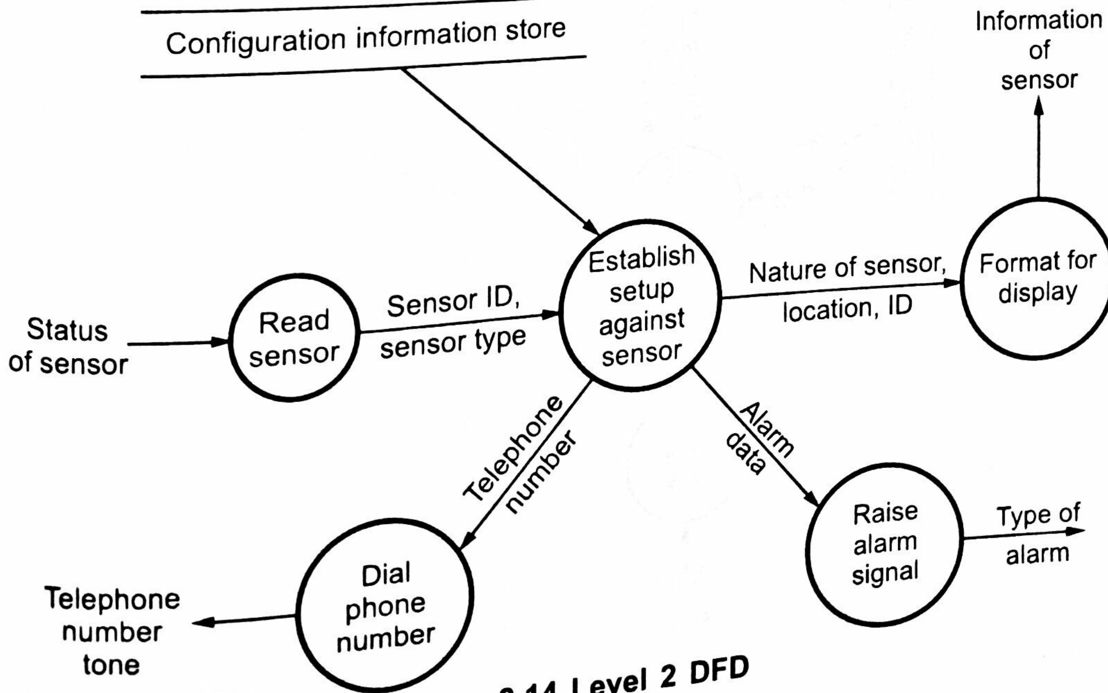


Fig. 8.14 Level 2 DFD

Step 3 : Determine if the DFD has the transform or transaction flow Characteristics

The information flow within the system is usually represented as transform flow. However, there can be dominance of transaction characteristics in the DFD. Based on characteristics of the DFD the transformation flow or transaction flow is decided.

For example if we draw a level 3 DFD for the process of 'Establish setup...' .The transformation flow is identified.

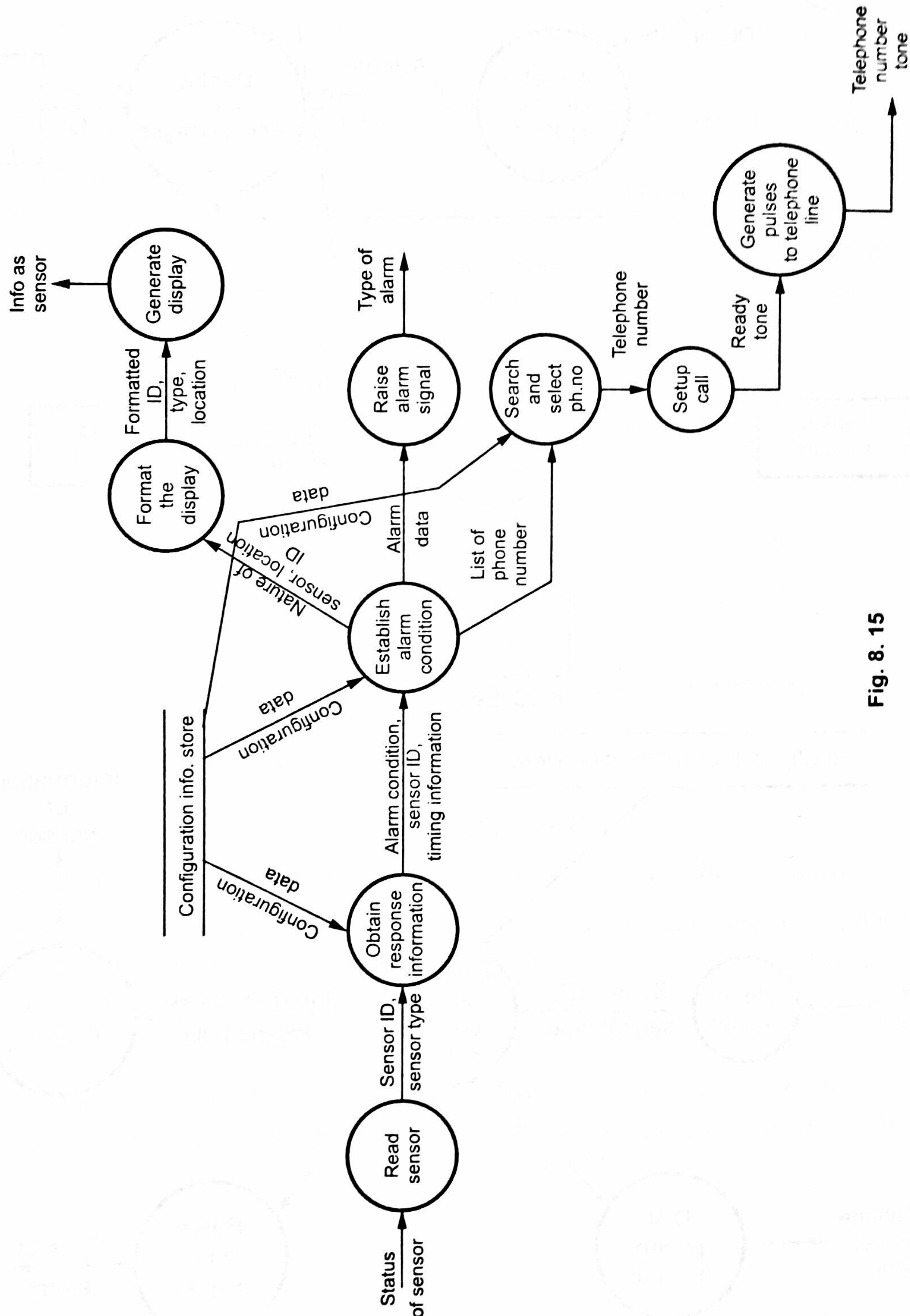


Fig. 8. 15

Step 4 : Isolate the transform centre by specifying incoming and outgoing flow boundaries

In this design step, the reasonable boundaries are selected and it is avoided to apply lengthy iterations on placement of divisions. The incoming and outgoing flow boundaries are as shown in Fig. 8.16.

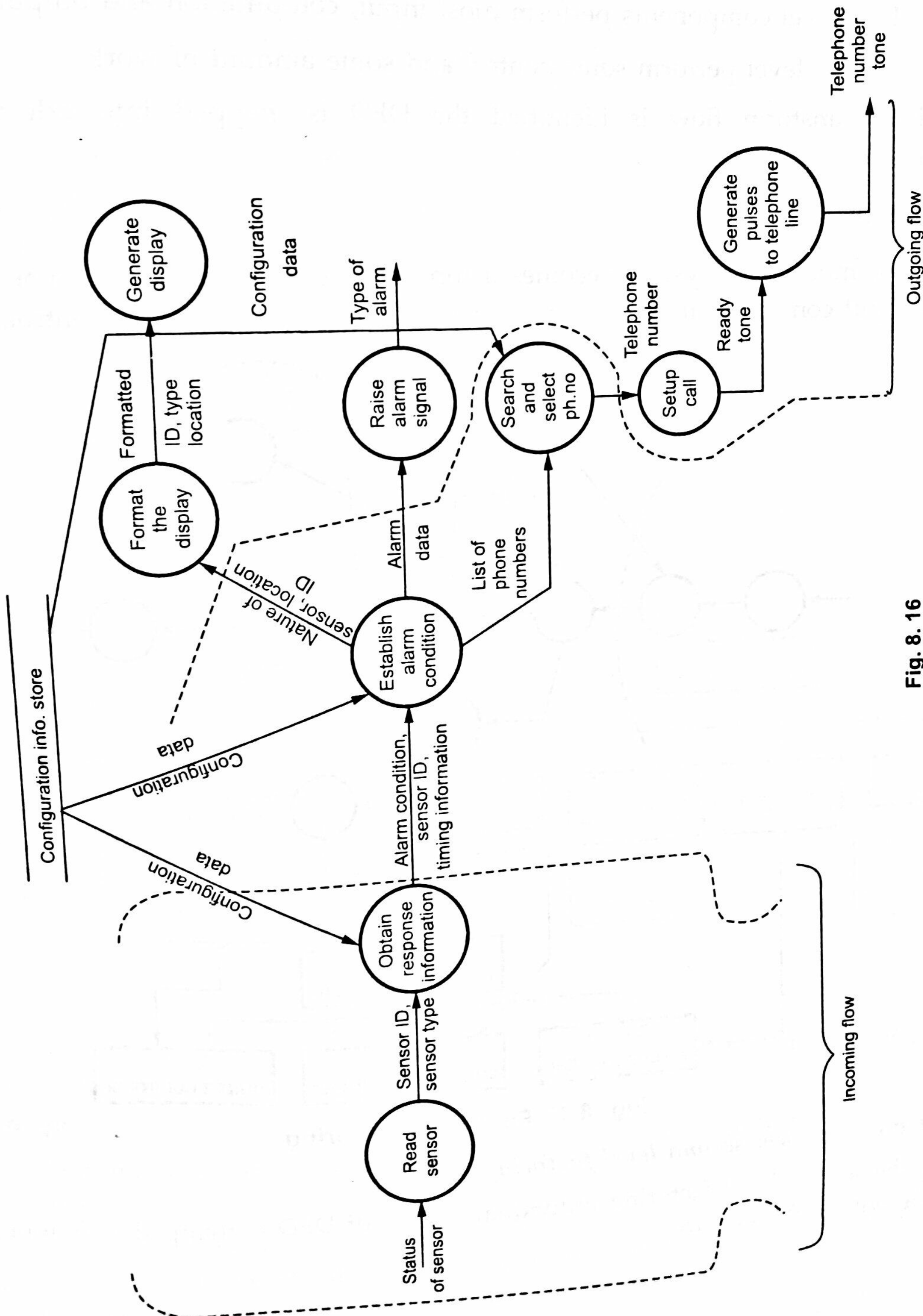


Fig. 8.16

Step 5 : Perform first level factoring

After performing the first level factoring the program structure results in top down architecture where

- Top level components perform decision making
- Low-level components perform most input, computation and output
- Middle-level perform some control and some amount of work

When transform flow is identified the DFD is mapped into call and return architecture.

For example

Sensor monitoring system becomes a top level component which co-ordinates the sensor input controller, alarm conditions controller and alarm output controller.

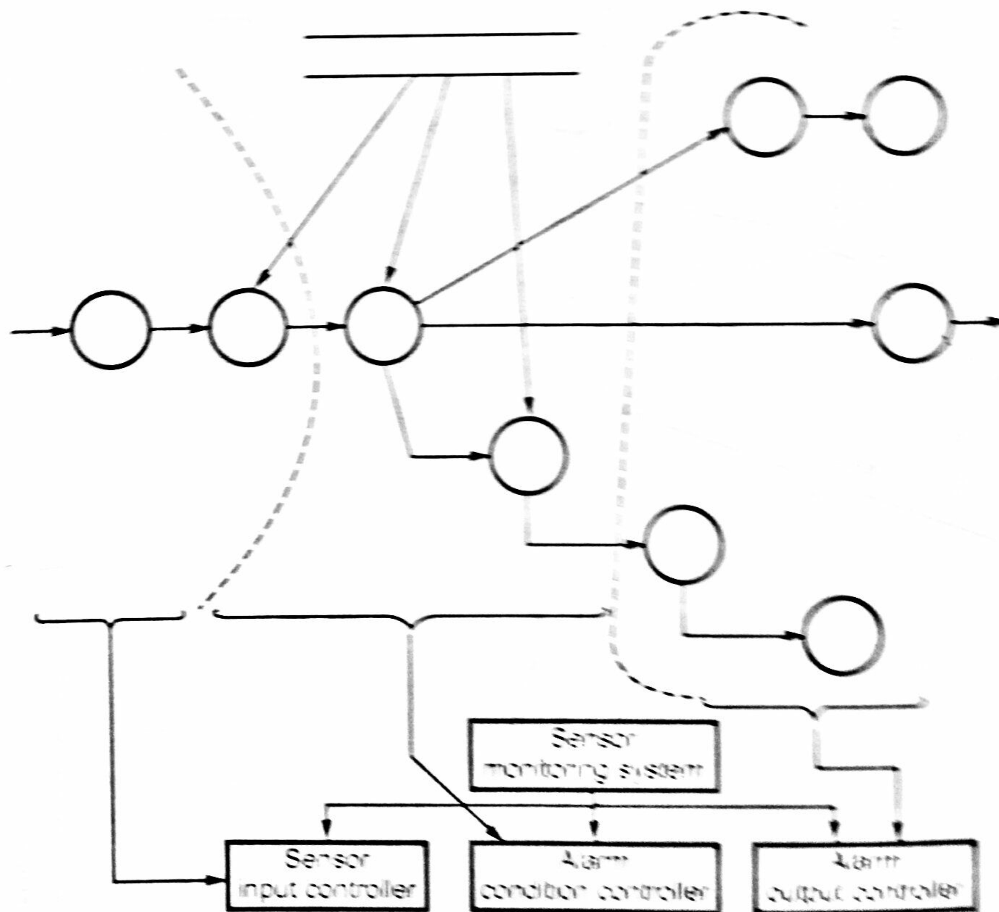


Fig. 8.17 First level factoring

Step 6 : Perform second level factoring

In the second level factoring individual bubble of DFD is mapped into appropriate module within architecture.

There could be one-to-one mapping of bubble of DFD into the software module or two or three bubbles can be combined together to form a single software module.

After performing the second level factoring the architecture serves as the first-iteration design.

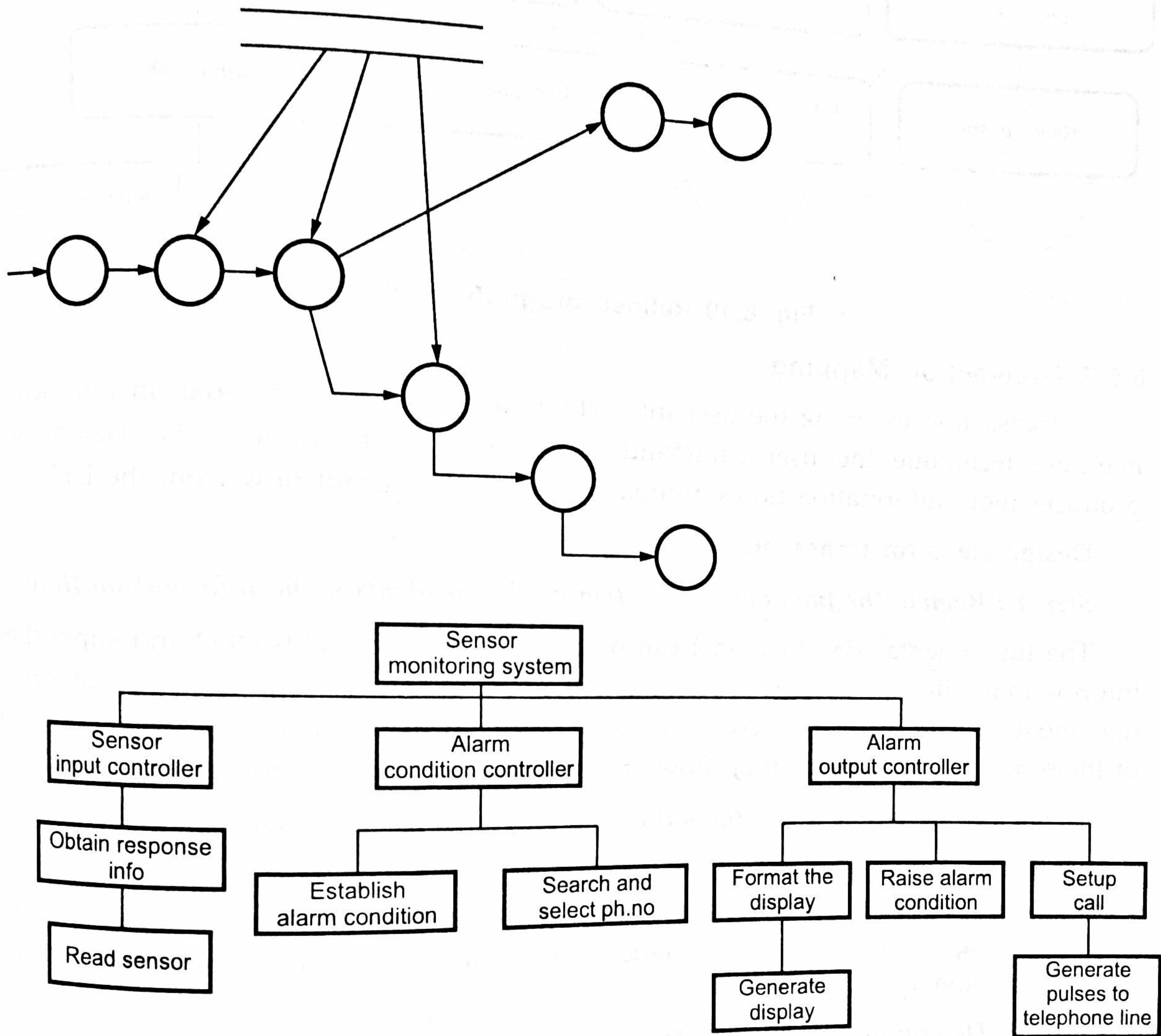


Fig. 8.18 Second level factoring

Step 7 : Refine the first-iteration architecture using design heuristics for improved software quality

The first-iteration architecture can be refined by applying the module independency.

The modules can be exploded or imploded with high cohesion and minimum coupling. The refinement should be such that the structure can be implemented without difficulty, tested without confusion and can be easily maintained.

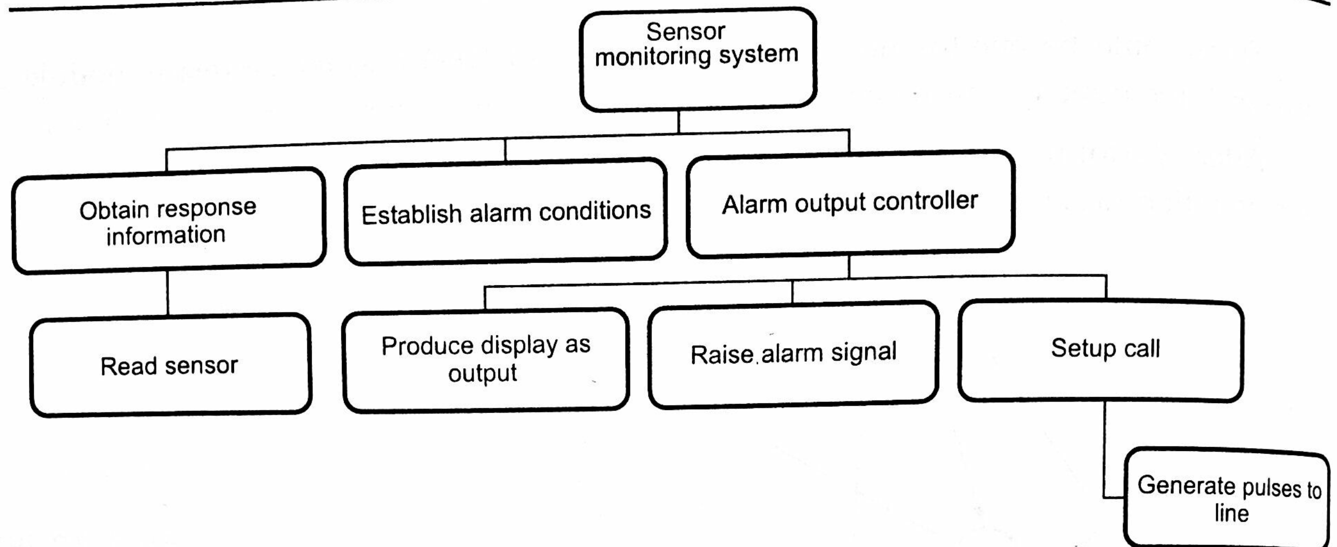


Fig. 8.19 Refined program structure

8.6.2 Transaction Mapping

In transaction mapping the user interaction subsystem is considered. In transaction mapping technique the user command given as input flows into the system and produces more information flows, ultimately causes the output flow from the DFD.

Design steps for transaction mapping

Step 1 : Review the fundamental system model to identify the information flow

The fundamental system model can be represented by level 0 DFD and supporting information. This supporting information can be obtained from the two important documents called 'system specification' and 'software requirement specifications'. Both of them describe the information flow and structure at software interface.

Step 2 : Review and refine the data flow diagrams for the software

The data flow diagrams are analyzed and refined into next higher levels. Each transform in the data flow diagrams impose relatively high level cohesion. That means after applying the certain transformation the process in the DFD performs a single distinct function.

Step 3 : Determine if the DFD has the transform or transaction flow Characteristics

The information flow within the system is usually represented as transform flow. However, there can be dominance of transaction characteristics in the DFD. Based on characteristics of the DFD the transformation flow or transaction flow is decided.

Step 4 : Identify the transaction centre and flow characteristics along each of the action paths

In transaction mapping we have to identify the location of transaction centre. From the transaction centre many action paths flow radially from it.

For example : As shown in following level 2 DFD the command processing centre serves as the transaction centre. The reception path and action paths are also shown .

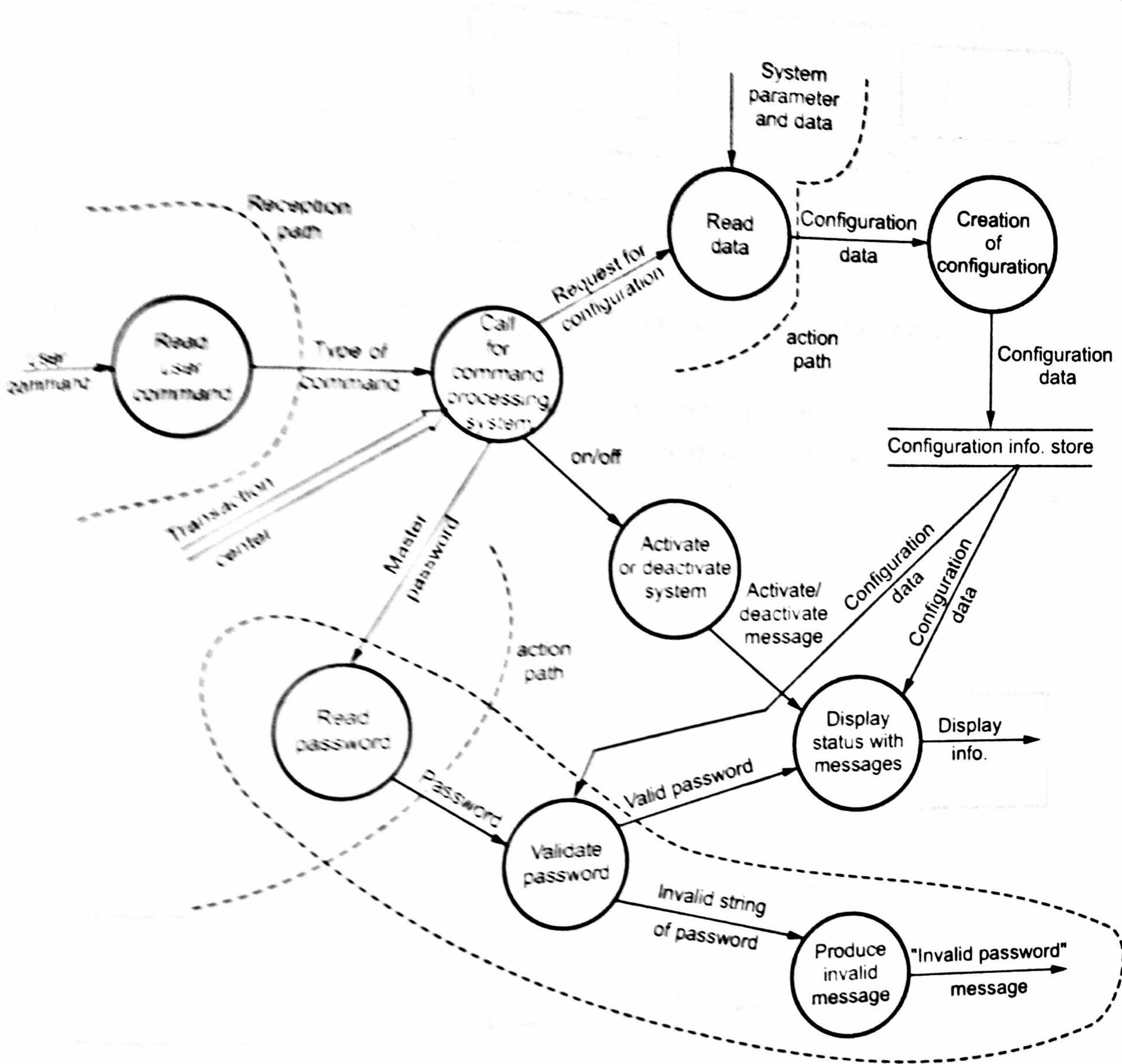


Fig. 8.20 Transaction centre

Step 5 : Map DFD into transaction processing structure

The identified transaction flow is mapped into an architecture that contains an incoming branch and a dispatcher branch. Starting from the transaction centre the corresponding bubbles on incoming path (path coming to transaction centre) are mapped into the appropriate modules. The bubbles along the action path are mapped into the action modules.

For example

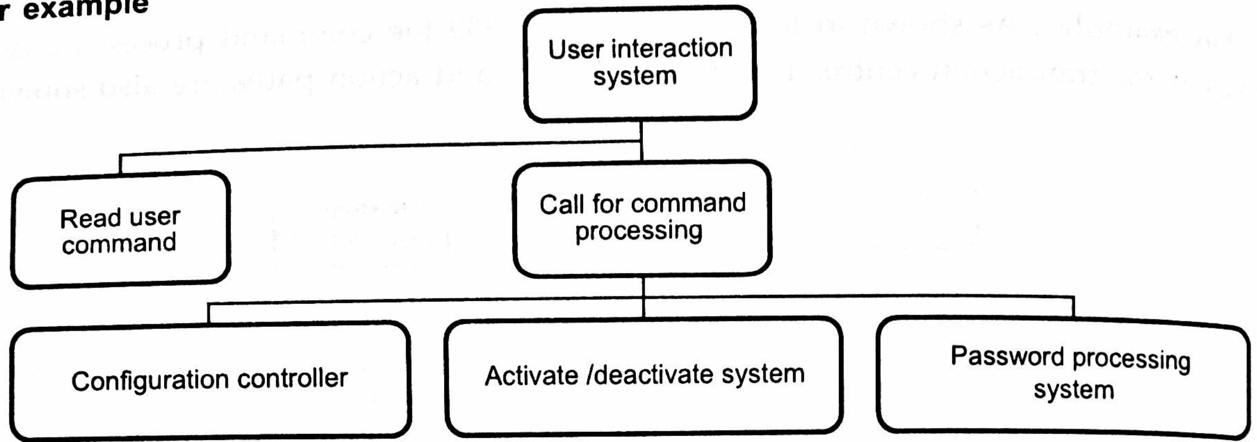


Fig. 8.21

Step 6 : Factor and refine the transaction structure and structure of each action path

Each action path of the data flow diagram has its own information flow characteristics. The action path related substructure is developed. This substructure serves as first iteration architecture. Refer Fig. 8.22.

Fig. 8.22 First iteration architecture

Step 7 : Refine the first-iteration architecture using design heuristics for improved software quality

The first-iteration architecture can be refined by applying the module independency.

The modules can be exploded or imploded with high cohesion and minimum coupling. The refinement should be such that the structure can be implemented without difficulty, tested without confusion and can be easily maintained.