UNIT II

CHAPTER 5

System Design

Syllabus

System/Software Design, Architectural Design, Low-Level Design Coupling and Cohesion, Functional-Oriented Versus The Object-Oriented Approach, Design Specifications, Verification for Design, Monitoring and Control for Design.

5.1 Introduction

This chapter focuses on the design phase of SDLC. After requirement gathering and analysis phase, the next step is to *design the analysis model*.

System

System is an interrelated set of components, with identifiable boundaries working together for achieving some goal.

A system has nine characteristics

1. **Components:** These are the indecomposable or irreducible parts that make up a system which are also called as subsystems.
2. **Inter-related Components:** It is about the dependency between the subsystems i.e. dependency of one subsystem on one or more subsystems.
3. **A boundary:** It is the line that marks the inside and outside of a system and that sets off the system from its environment.
4. **A purpose:** It is the overall goal or function of a system.
5. **An environment:** It encompasses all the external entities of a system that interact with the system.
6. **Interfaces:** Interaction between the system and its environment or interaction between various subsystems.
7. **Input :** Whatever a system takes from its environment in order to fulfil its purpose.
8. **Output:** Whatever a system returns to its environment in order to fulfil its purpose.
9. **Constraints:** The limit to which a system can accomplish or the restrictions on developing the system.

Syllabus Topic : System/Software Design

5.2 System Design

Systems analysis : Process of studying an existing system to determine how it works and how it meets user needs.

Systems design : Process of designing a plan for an improved system based upon the results of the existing system analysis.

Need of System Design

- It is an Effective method to solve problems of existing system as it designs the new plan based upon the study of existing system analysis.
- It differentiates logical view and physical view of the software to be built up.
- Designing also partitions the requirements such that it becomes easy to build up a model and develop its documentation.
- It keeps track of all interfaces and designs them properly.
- Whenever possible, graphics are used to design the analysis model.

Main Objectives of Design

- Practicality – Efficiency – Cost – Flexibility – Security

Analysis & Design Model

- Analysis modelling uses diagrammatic form and text to describe requirements of :
  - Data o Functions o Behaviour of o The software to be built
- It should have the following features :
  - Easy to understand. o Straightforward to review. o Correctness.
  - Completeness. o Consistency.

Who does it ?

- Software engineers or – System analysts or – Project manager or – Modeller

Importance

- Analysis model is multidimensional.
- Design phase completely depends on analysis model.
- If some deficiency remains in the analysis models then errors will be found in product to be built.

Ensures

- It must be review for correctness, completeness and consistency.
- It should ensure that it accomplishes all needs of stakeholders.
- It establishes a foundation from which design can be conducted.

5.2.1 Thumb Rule

Arlow and Neustadt produce rules of thumbs for the analysis model. These rules are given below:

1. Model should focus on requirement visible in business domain.
2. Each element of the analysis model should add to all requirements and provides insight into information domain function and behaviour of the system.
3. Some functional and non functional models are required to design the software.
4. Minimize coupling throughout the system. It is important to represent relationship between classes and functions. That means functions are interconnected to each other very tightly then efforts are put to reduce this interconnectedness.
5. It gives assurance that analysis model provides value to all stack holders.
6. It keeps model as simple as it can be.

5.2.2 Design Modeling Principles

Design model provides a variety of different views of the system just like architecture's plan for a house. Different methods like data driven, pattern driven or object oriented methods are used for constructing the design model. And all these methods use a set of design principles for designing a model.

1. **Design must be traceable to the analysis model** Analysis model represents the information, functions and behaviour of the system. Design model translates all these things into architecture: a set of sub systems that implement major functions and a set of component level designs that are the realization of analysis classes. This implies that design model must be traceable to the analysis model.
2. **Always consider architecture of the system to be built** Software architecture is the skeleton of the system to be built. It affects interfaces, data structures, program control flow and behaviour, the manner in which testing is conducted, maintainability of the resultant system and much more.
3. **Focus on the design of data :** Data design encompasses the manner in which the data objects are realized within the design. It helps to simplify the program flow, makes the design and implementation of software components easier and makes overall processing more efficient.
4. **Interfaces (both user and internal) must be designed :** The data flow between the components decides the processing efficiency, error flow and design simplicity. A well designed interface makes integration easier and tester can validate the component functions more easily.
5. **User interface should consider the user first :** The user interface is the main thing of any software. No matter how good its internal functions are or how well designed its architecture is but if the user interface is poor and end users don't feel ease to handle the software then it leads to the opinion that the software is 'bad'.
6. **Component level design should exhibit functional independence :** It means that the functions delivered by a component should be cohesive i.e. it should focus on one and only one function or sub function.
7. **Components should be loosely coupled :** Coupling (Union or Combination) of different components into one is done in many ways – via a component interface, by messaging or through global data. As the level of coupling increases, error propagation also increases and overall maintainability of the software decreases. Thus, component coupling should be kept as low as possible.
8. **Design representations should be easily understood :** The design model helps engineers to generate code, testers to test software, and maintain the software easily in the future. If the design is difficult to understand, it will not serve as an effective communication medium.
9. **The design model should be developed iteratively :** Designs are done in iterations to make designing as simple as possible.

Syllabus Topic : Architectural Design

5.3 Architectural Design (High-level Design)

Architecture design of a system or a software constitutes of a complete framework that includes the structure – its components and how they are integrated with each other to form a complete system.

Need of Architectural design

- It evaluates all top-level designs.
- Highlights early design decisions by specifying the functional and performance behaviour of the system that leads to ultimate success of the system
- It constitutes of a relatively small and easily understandable model of how the system is structured and how its components are integrated to work together. This enables communication between all stakeholders of the system

Architectural Design Representations

1. **Structural model :** represents the ordered collection of program components
2. **Dynamic model :** shows the behavioral aspect of the software and indicates how the system configuration changes with the change in the functions.
3. **Process model :** depicts the design of the business or technical process that needs to be implemented in the system

Functional model: Represents the functional hierarchy of a system

Framework model: shows increase in the level of abstraction.

Outputs of Architectural Design

1. **Reports :** audit report, progress report, and configuration status accounts report
2. **Plans :**
   i) Software verification and validation plan
   ii) Software configuration management plan
   iii) Software quality assurance plan
   iv) Software project management plan.

Architectural Styles

1. Data centered architecture

This type of architecture has two distinct components - a central data structure known as data store (central repository) and a collection of client software.

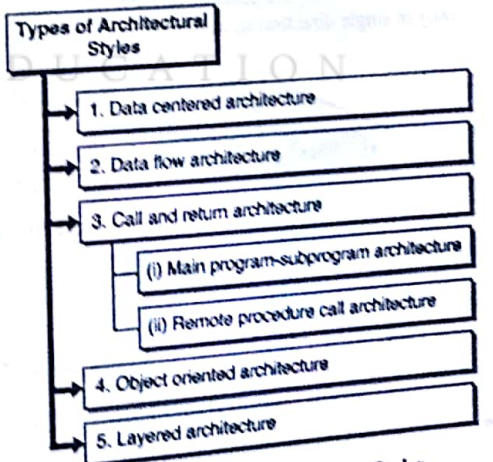


Fig. C5.1 : Types of Architectural Styles

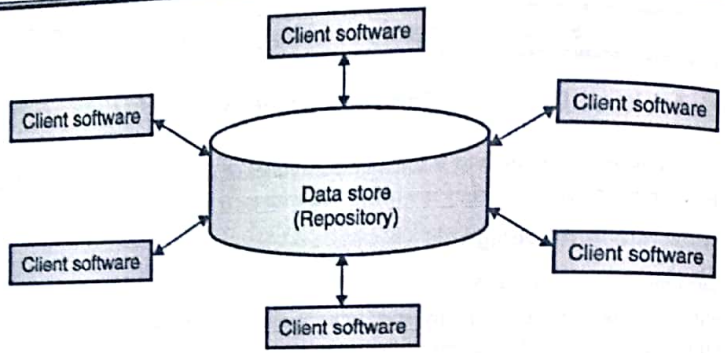


Fig. 5.3.1 : Data centred architecture

Advantages

- Client software operate independent of each other
- Central repository is independent of client software
- Scalable : new clients can be added easily
- Easily modifiable

2. Data flow architecture

This type of architecture is used for the systems that accept inputs and transform it into the desired outputs by applying a series of transformations. Each component called as filter transforms the data and sends it to other components (filters) for further processing using the connector known as pipe. Each filter works independent of each other and each pipe is a unidirectional channel that works only in single direction so as to pass the data from one filter to another filter.

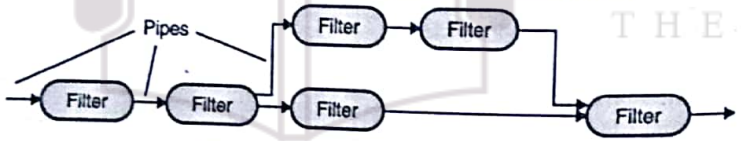


Fig. 5.3.2 : Data flow architecture

Sr. No.	Advantages	Drawbacks
1.	Supports reusability	Does not provide enough support for applications requiring user interaction.
2.	Maintainable and modifiable Supports concurrent execution	Difficult to synchronize two different streams

3. Call and return architecture

This type of architecture designs a program structure that can be easily modified. This architecture consists of following two sub styles.

(i) Main program-subprogram architecture:

The main function is decomposed to invoke number of program components which in turn invoke other components.

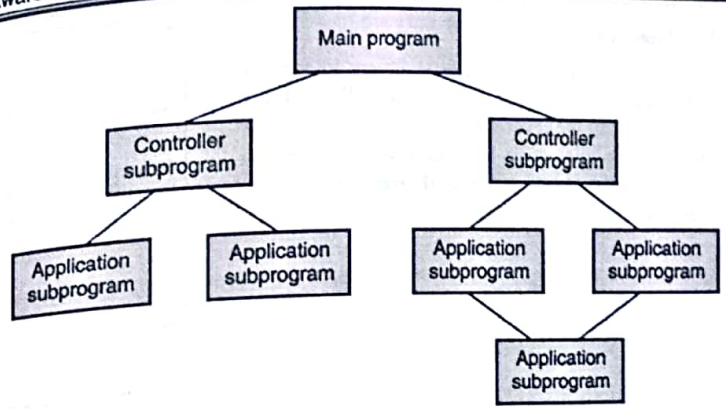


Fig. 5.3.3 : Main program-subprogram architecture

(ii) Remote procedure call architecture:

Components of the main program or subprogram architecture are distributed over a network across multiple computers.

4. Object oriented architecture

This type of architecture designs the components known as **objects** of the system such that it encapsulates data and operations (used to manipulate the data) of the object. In this style, the objects interact with each other through methods known as **connectors.**

Advantages

- Maintains the integrity of the system
- Allows to decompose the problem into a collection of independent objects
- Encapsulation allows to hide the implementation details of one object from other objects. Therefore, every object can be changed without effecting other objects.

Object Oriented architecture constitutes of 4 types of modelling elements : 1

- Scenario based elements.
- Flow oriented elements.
- Class based elements.
- Behavioural elements.

The specification of each element may differ from project to project.

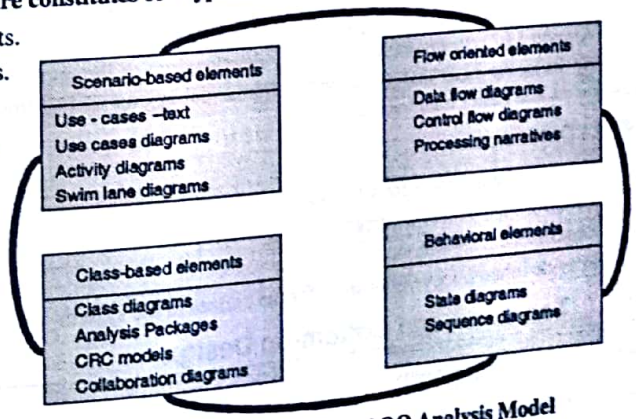


Fig 5.3.4: Different types of OO Analysis Model

5. Layered architecture

This type of architecture consists of several layers known as components arranged in a hierarchical manner. Each layer provides a set of services to the layer above it and acts as a client to the layer below it. The interaction between layers is provided through protocols known as connectors that define a set of rules to be followed during interaction.

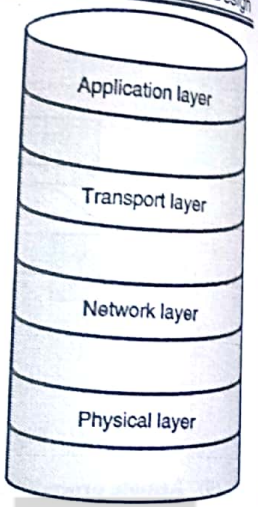


Fig. 5.3.5: Example of Layered architecture – OSI layered architecture

Syllabus Topic : Low-Level Design

5.4 Low-Level Design

- High-level design (HLD) identifies the system's general environment (hardware, operating system, network, and so on) and architecture (client/server, and service-oriented). It works on modules such as reporting modules, databases, and top-level classes.
- Low-level design (LLD) provides extra details that are necessary before developers can start writing code. It tells how the parts of the system will work together and refines the definitions internal and external interfaces.
- High-level designs focus on *what*. Low-level designs focus on *how*.
- HLD designs the overall architecture of the entire system from main module to all sub module. LLD defines Internal logic of corresponding sub modules.

A good LLD document helps in easy development of the application. The code can be developed directly from the LLD document with minimal debugging and testing. Other advantages of LLD are lower cost and easier maintenance.

LLD document

- Provides the interface for all classes
- Describes the data structures
- Describes the algorithms
- Provide inheritance relationships for the classes in the system

5.5 Top-down and Bottom-up Design

Top-Down Approach

- It is also known as step-wise design and is about breaking down of a system to get details of its compositional sub-systems.

- In a top-down approach, the overview of the system is first formulated which specifies but does not give any detailing of first-level subsystems. Each subsystem is then refined in yet greater detail, sometimes in many additional subsystem levels, until the entire specification is reduced to base elements.
- A top-down model is usually specified using the 'black boxes' which make it easier to manipulate.
- Top-down approach focuses on planning and a complete understanding of the system. It strictly tells that no coding can begin until a sufficient level of detail has been reached in the system design of at least some part of the system.

Advantages of Top-Down Approach

- Separating the low level work from the higher level abstractions leads to a modular design where development of sub systems become self contained i.e. independent of each other.
- Illustrates the integration of low level modules.
- Reduces errors because each module has to be processed separately, so programmers get lot of time for processing.
- It is less time consuming because each programmer is only involved in one part of the big project.
- Each programmer has to apply his knowledge and experience to only his module and so the project becomes more optimized.
- Easy to maintain because if an error occurs in the output, it is easy to identify the source of the error i.e. it is easy to find that which module of the entire program has caused that error.

Bottom-Up Approach

- It is about bringing together of sub systems to give rise to a better system, thus making the original systems as the sub-systems of the emergent system.
- In a bottom-up approach, individual base elements of the system are first specified in great detail. These elements are then integrated together to form larger subsystems, which then in turn are linked, sometimes in many levels, until a complete top-level system is formed.
- Bottom-up emphasizes coding and early testing, which can begin as soon as the first module has been specified. This approach, however, runs the risk that modules may be coded without having a clear idea of how they link to other parts of the system.

Syllabus Topic : Functional-Oriented Versus The Object Oriented-Design Approach

5.6 Function Oriented vs. Object Oriented Design

- Jim Rumbaugh and his Co-workers found out the method of the Object Modeling Technique (OMT) which describes the analysis, design and implementation by using OMT.
- Rumbaugh method further became very popular among various authors because of its user friendly nature and simplicity ways.
- Rumbaugh uses familiar symbols and techniques in his analysis and design.

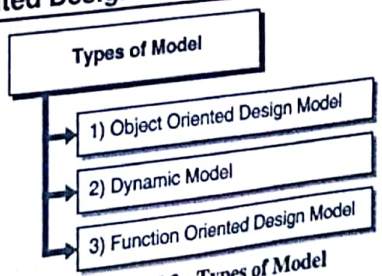


Fig. C5.2 : Types of Model

- Rumbaugh OMT provided three set of concepts which provide three different views of the system. The three models are as define below;

1) Object Oriented Design Model

Object Model describes the static structure of the objects in the system - identification of attributes, operations and their relationship.

The main concepts are;

- Class
- Attributes
- Operation
- Inheritance
- Association (i.e. relationship)
- Aggregation

- The Rumbaugh object model is very much like an entity relationship diagrams (ERD) except that there are no behaviors in the diagram and class hierarchies.
- Following are the notations used by the Rumbaugh method in his object model;

Table 5.6.1 : Notation used by Rumbaugh in his Object model

Sr. No.	Shape Used	Purpose
1	Class / Attributes / Operation	Class Representation
2.		One-to-one association
3		One-to-many association
4		Specialisation

Example

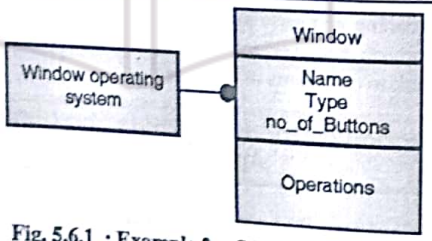


Fig. 5.6.1 : Example for OMT object Model

- OOD designs technical solutions (User Interface, Data Schemas, Object Classes, etc) to the problems identified by analysis phase.
- When we are solving problems using OOD, the following tasks must be accomplished by the user.
  - Basic user requirements are known to customer as well as software engineer
  - Design the classes according to requirements –Attributes and Methods are defined.
  - Hierarchy of the classes are defined.
  - Relationships among classes are defined.
  - Behaviour of object is decided.

  - All above tasks are repeated till model is not complete.

2) Dynamic Model

- The dynamic model is a "state transition" diagram that shows how an entity changes from one state to another state.
- The main aspects are;
  - State
  - Sub / Super State
  - Event
  - Action
  - Activity
- This kind of diagram states network of activity.
- Following are the notations which are used by the Rumbaugh method in his dynamic model;

Table 5.6.2 : Notation used by Rumbaugh in his Dynamic model

Sr. No.	Shape Used	Purpose
1		Start
2.		States
3		Transition

Example

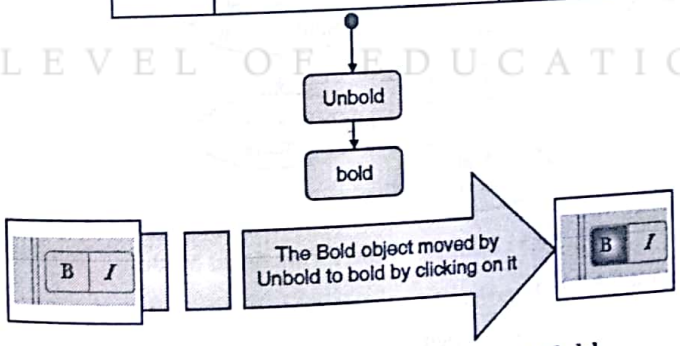


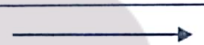
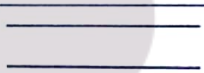
Fig. 5.6.2 : Example of OMT Dynamic Model

3) Function Oriented Design Model

- The functional model is the equivalent of the familiar Data Flow Diagrams (DFD) from a traditional systems analysis.
- This model describes the data value transformation within the system.
- A *function model* in systems engineering and software engineering is a structured representation of functions, behaviors, activities or processes within the modeled system.
- A function model, also called an activity model or process model, is a graphical representation of an enterprise's function within a defined scope.

- A well-known example of functional modeling language is data flow diagrams (DFD).
- The main concepts are :
  - Process
  - Data Store
  - Data Flow
  - Control Flow
  - Actor (Source / Sink)
- Following are the notations which are used by the Rumbaugh method in his functional model;

Table 5.6.3 : Notations used by Rumbaugh in his Function model

Sr. No.	Shape Used	Purpose
1.		Process
2.		Data Flow
3.		Data Store
4.		External Entity

Example

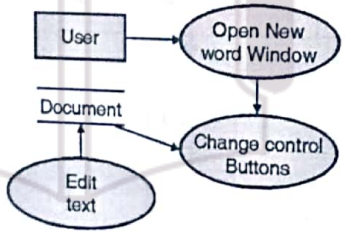


Fig. 5.6.3 : Example of OMT Functional Model

Table 5.6.4 : Function Oriented v/s Object Oriented Design

Sr. No.	Function Oriented Design	Object Oriented Design
1.	Focuses on functions which operate on data.	Focuses on objects which are conceptual entities having attributes and methods.
2.	It is a set of functions each of which performs a task.	It is a set of objects that can be modified and that can perform tasks.
3.	Relies on identifying functions which transform their inputs to create outputs.	Relies on encapsulation of objects and messaging between objects.

Sr. No.	Function Oriented Design	Object Oriented Design
4.	Divides a bigger problem set to small functional units and then organizes these functional units to design the solution.	Identifies the objects involved in the system and designs the solution based on their relationships and interactions.
5.	Used mainly for computation of sensitive applications.	Used mainly for evolving systems that mimic a business process.

Syllabus Topic : Design Specification

5.7 Design Specification

- Design specification enables to make planned decisions with broad consequences.
- Best design specification is implemented by user requirements as well as previous experiments.
- The design process starts from object oriented Analysis and ends with system design.
- Design specifications can be categorized into two broad categories .

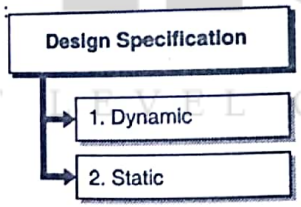


Fig. C5.3 : Design Specification

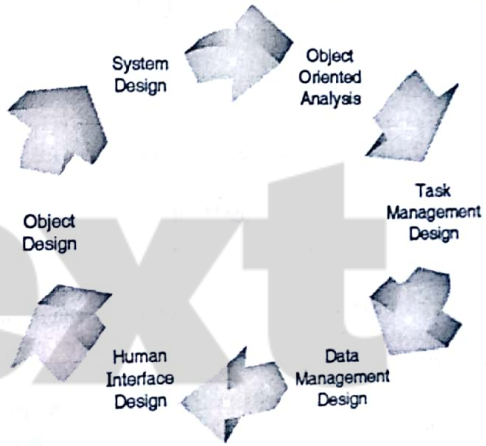


Fig. 5.7.1 : Process Flow of a Design

➜ **1. Dynamic**

- Data flow diagrams (DFDs)
- State transition diagrams
- Statecharts
- Structure diagrams

➜ **2. Static**

- Entity Relationship Diagrams(ERDs)
- Class diagrams
- Structure charts
- Object diagrams

- The design specification helps developers to take decision on how the problem will be solved.
- During this system design stage, it solves the problems which basically arises from the system analysis phase where developers think on :
  - Overall structure of the system
  - Subsystem of the system
  - Overall style of the system
  - Reuse of the system

5.8 Partitioning the Model

- The partitioning of the target system into subsystems, based on the combination of knowledge of the problem domain and the proposed architecture of the target system (solution domain).
- The breaking up of the system into the small pieces of the subsystems is the first step of the system design.
- Since the major piece which further implement is nothing but the sub system. That's nothing but the partitioning of the model.

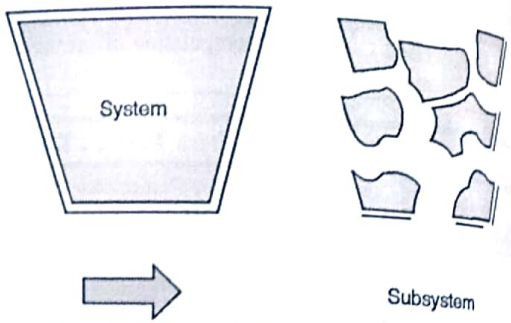


Fig. 5.8.1 : Partitioning the Analysis Model

- A subsystem is not an object nor a function but a group of classes, associations, operations, events and constraints that are interrelated and have a well defined and small interface with other subsystems.
- Whatever services provided by the subsystem according to that subsystems are identified.
- The subsystem should have a well-defined interface through which all communication with the rest of the system occurs.
- With the exception of a small number of "communication classes," the classes within a subsystem should collaborate only with other classes within the subsystem.
- The number of subsystems should be kept small. Subsystem may in turn be decomposed into smaller subsystem, of its own. The lowest level subsystems are called as modules.
- The decomposition of system into subsystems may be organized as a sequence of horizontal layers or vertical partitions.

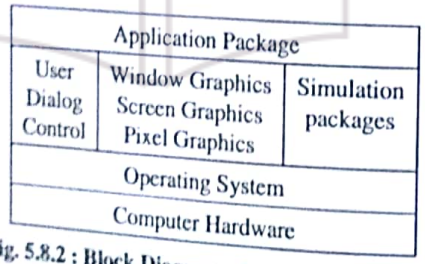


Fig. 5.8.2 : Block Diagram of typical application

- A subsystem can be partitioned internally to reduce complexity.

5.9 Abstraction and its Types

- Abstraction can be defined as an ability to look at something without being concerned with its internal details.
- This concept was introduced with structured programming where it is sufficient to know that a given procedure performs a specific task or not.

Abstraction Types

There are two types of Abstraction –

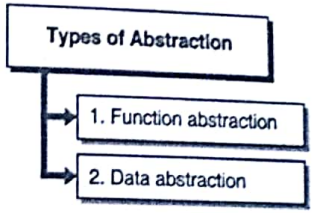


Fig. C5.4 : Types of Abstraction

1. Function abstraction

How is it done is not at all important as long as the procedure is reliable. This is known as ***Function abstraction***. For example, when you are using printf() function in C you are not concerned about how it is implemented but you know that it can be reliably used to output the data on the screen.

2. Data abstraction

It is used for data as functional abstraction for operations. With data abstraction, data structure and items can be used without having to be concerned about the extract details of implementation. For example, floating point numbers are abstracted in programming language.

- It means that one need not bother about the binary representation of floating point numbers gauges. It means that one need to know the details of binary multiplications in order to multiply two floating-point numbers.
- Similarly, a data type called fraction can be implemented using data abstraction.
- Since the classes use the concept of data abstraction, they are known as Abstract Data Types (ADT).
- We see that a tablet capsule is made up of two or more chemical materials. That chemical material is encapsulated in a cover. The same feature is adapted in OOD.
- Wrapping of data and functions into single unit (or class) is known as 'Encapsulation'. Data encapsulation is the most salient features of a class.
- The data is not accessible to the outside world. It is accessible by only those functions, which are wrapped in a class.

5.10 Modularity

- Modularity is about dividing the complex system into modules or components. The software is decomposed into separately named and addressable components.
- Modularity must be effective i.e. each module must focus on exclusively well-constrained aspects of the system i.e. it should be cohesive in its functions.
- Also, modules should be interconnected in a relatively simpler manner i.e. each module should show low coupling with other modules.
- A **module** is a system component that provides services to other components and is itself dependent on the services provided by other components. It is not normally considered as a separate system.

A system can be divided into different categories of modules or components :

Problem domain component

- The subsystem that is responsible for implementing customer requirements directly.
- Depending on the customer requirements, the design pattern knows the problem exactly.

Human interaction component

- This is a very important component of the OO design model since using this component, a model interacts with the Human.
- The subsystems that implement the user interface (this includes reusable GUI subsystems).
- For example Microsoft Excel has various cells in which we put entries and manipulate them. Each different entry is classified into different manner.

Table 5.10.1 : Example of Microsoft Excel Database

ID	Name	Salary
1	Nilesh	8000
2	Atul	5700
3	Sourabh	7000
4	Sunil	10000
		30700

- In Table 5.10.1 we make three columns ID, Name and Salary which are according to the class design. Here all of the three are attributes, Operations(Here we make sum of third column Salary) we use *sum* method for that.
- We also use alternative pattern in the OO Design.

Task management component

The subsystems that are responsible for controlling and coordinating concurrent tasks that may be packaged within a subsystem or among different subsystems;

Data management component

The subsystem that is responsible for the storage and retrieval of objects.

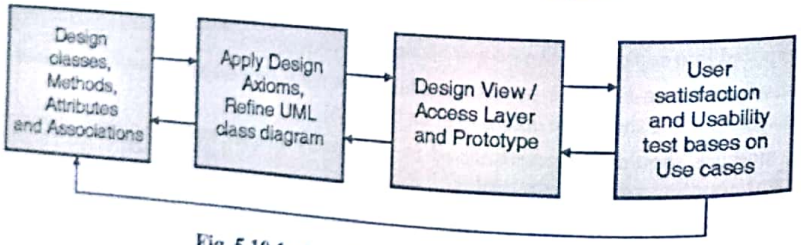


Fig. 5.10.1 : Looping within components

Advantages of Modularity

- By modularizing the system design, planning the development becomes more easy.
- Modules can then be developed in increments and delivered fast.
- Changes to modules can be easily accommodated as the changes in one module don't affect the other.

- Testing and debugging can be performed more effectively.
- Long-term maintenance can be performed by no serious side effects.

Syllabus Topic : Coupling and Cohesion

5.11 Coupling and Cohesion

Coupling

- Coupling within a software system is the degree to which each module depends on other modules. It is the degree of dependency among the components.

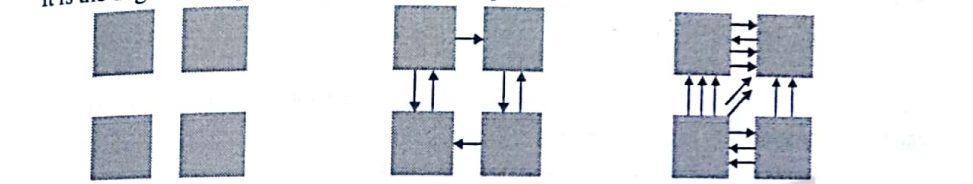


Fig. 5.11.1 : Various Types of Coupling

- Coupling is generally considered a bad thing where software components are dependent for some specific details of other components.
- In high coupling, the components are very closely and tightly tied to each other which make it difficult to modify the components of the system because modifying a component affects all other components to which the modifying component is connected.
- ***Therefore, Low-Coupling is good.***
- Object-oriented design has low coupling i.e. the objects (components) are independent of each other. Therefore, modifying one object doesn't have any effect on another object.

Example : Functions used in C++ & Java programming represent coupling i.e. dependency between software components (functions)

Cohesion

- Cohesion is the measure of internal interdependence with in a component i.e. the degree to which all elements of a component are directed towards a single task is called cohesion.
- ***Therefore, High-Cohesion is good.***
- Cohesion is considered as good thing because it is about creating a software component that combines related functionalities into a single unit. This is the core principle of OOD where it creates a component that encapsulates the objects of similar properties and functions that strive to perform same single task.

Example : Classes and objects in C++ & Java programming represent cohesion i.e. the degree to which the elements of a single component form a meaningful unit (object) and perform similar task.

- Cohesion represents how tightly the internal elements of a module are bound to one another - Coupling is degree of independence between different modules.
- High cohesion and low coupling is main criteria for good s/w design.

5.12 Structure Charts

- A typical Structure chart is a top-down modular approach showing the functional hierarchy between the components. In addition, it also shows the *data* interfaces between the components.
- Structure chart is a graphical way to represent module decomposition hierarchy. They are dynamic models like the DFDs showing that show how one function calls the others.
- A structure chart is used as a design tool which supports in dividing and conquering a large software problem by breaking down a problem into modules that are small enough to be easily understood. The process is also called as top-down design or functional decomposition.

☞ Notations used in Structure Chart

A Structure chart shows the breakdown of a system to its lowest manageable levels arranged in a form of a tree.

- Each *module* is represented by a *rectangular box* containing the module name. The tree structure signifies the relationships between modules.
- The module *hierarchy* is displayed by linking rectangles with *lines*.
- *Inputs* and *outputs* are indicated with *arrows* - An arrow entering the box is its input and that is leaving the box is its output.
- *Data stores* are shown as *rounded rectangles* and user inputs as circles.

Example : Structure Chart for 'Paying Bill'

This structure chart represents data passing between modules. When the module 'Pay Purchase Bill' is executed, the pseudo code checks if the bill is already paid or not by searching for the payment receipt by executing 'Search Receipt' module and if the receipt is not found then it only it will execute the module 'Give Money to money Collector' and then the job is finished.

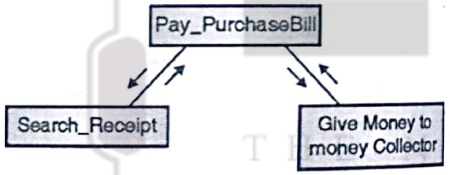


Fig. 5.12.1 : Structure Chart for 'Paying Bill'

☞ Steps to draw Structure Chart

Step 1: Identify the system processes: These processes are responsible for central processing functions that are not concerned with any input or output functions such as reading or writing data or data validation. Group such processes under a single function at the first-level in the structure chart.

Step 2: Identify input transformations: These are concerned with reading data or validating it and so on. Group such processes also under a single function at the first-level in the structure chart.

Step 3: Identify output transformations: These are concerned with preparing and formatting output and write it to the user's screen or other output devices.

☞ Structure chart is referred to as 'the master-plan' that signifies

- Size and complexity of the system number of easily identifiable functions and modules whether each identifiable function is a manageable module or should be broken down into smaller modules.

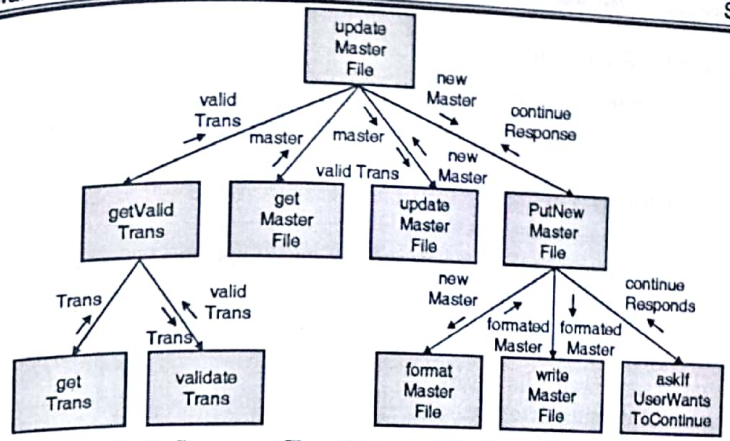


Fig. 5.12.2 : Structure Chart for 'Updating the Master File' function

5.13 Flow Charts

Flowchart is a modelling technique used in structured development and business modelling.

Flowchart is a graphical representation of the procedural logic of a computer program.

Example : Flowchart for 'Enrolment in University Course'.

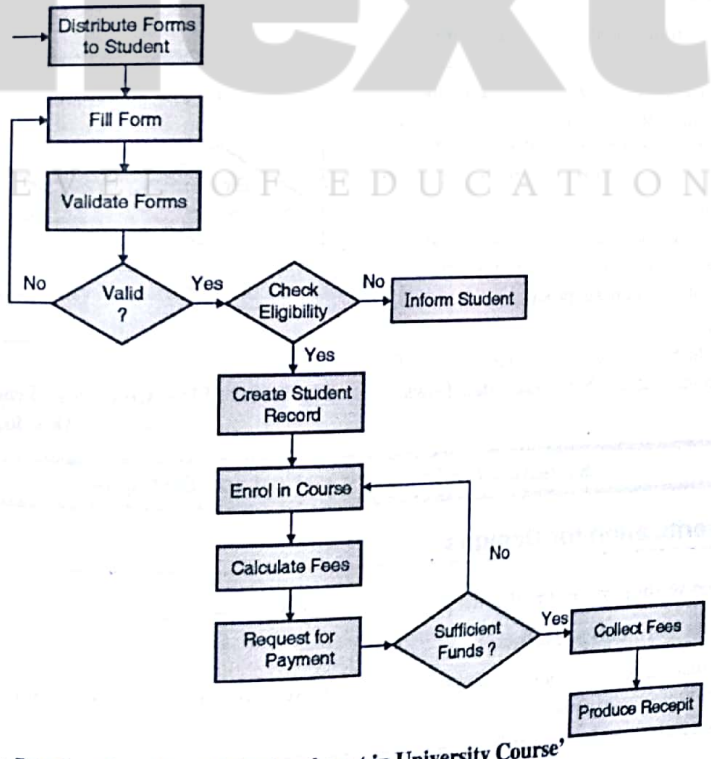


Fig. 5.13.1 : Flow Chart for 'Enrolment in University Course'

Notations used in Flow Chart

- *Rectangular boxes* represent the *activities* or tasks that are the computer instructions.
- *Diamonds* represent *decision points* i.e. a binary decision because there are two arrows leaving out from a decision.
- *Arrows* represent *flow of control.*
- Off-page connectors when diagrams get too big
- Input/output symbols represent printed reports and data storage options.

Example : Flowchart for 'Traditional Software Development'

Drawbacks

- Unless we take care, drawing flowcharts can be complicated and difficult to read.
- If the user's policy changes or if the systems analyst changes it several times before the user accepts it as correct, it will be time consuming and difficult job to redraw the flowchart each time.

Differences between Flowchart and the DFD

- DFDs show that all the bubbles (processes) in the system can be active simultaneously but the flow charts show individual process specification in a one-dimensional form i.e. the logic of the system flows uniformly from top to bottom.
- DFDs describe data flow within a system whereas, flow charts describe the detailed logic of a business process or business policy.
- Flowcharts represent sequencing of operations and DFDs represent data flows.

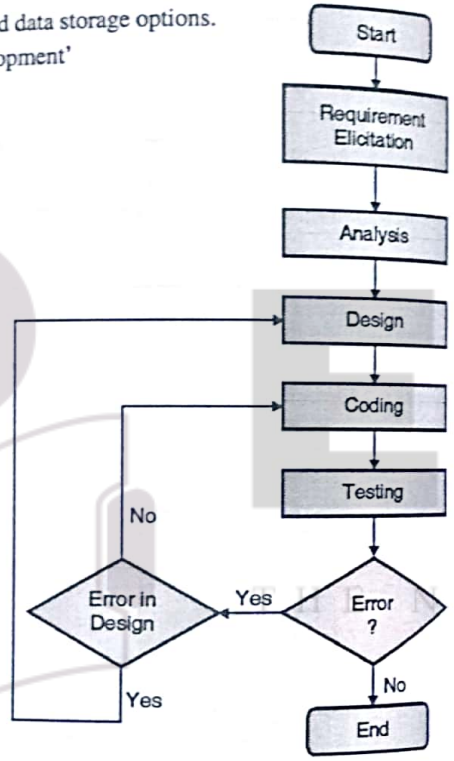


Fig. 5.13.2 : Flow Chart for 'Traditional Software Development'

Syllabus Topic : Verification for Designs

5.14 Verification for Designs

Design verification ensures that the product i.e. designed is the same as the product i.e. intended.

Design Verification Methods

1. **Demonstration :** It is done in actual or simulated environments. The cost of demonstration varies as per the complexity of the demonstration.
2. **Inspection :** It is done to verify the requirements related to physical characteristics.

3. **Analysis :** It is done to verify the design and is the most preferred method incase if testing is not feasible or incase when there is minimal risk.
4. **Testing :** It is the most expensive verification methods due to the project complexity as well as equipment and facility requirements. The most common method for validating this requirement is transportation testing.

Verification Activities

Fig. C5.5 : Verification Activities

1. Identification and preparation

- While developing a specification, the test engineer starts writing detailed test plan and procedures.
- Identifying the best approach of verification, identifying the measurement methods and required resources, tools and facilities.
- Once the verification plan is ready, it is reviewed by the design team to identify issues before finalizing the plan.

2. Planning

- Planning for verification is a simultaneous activity with core development activity which continues throughout the project life cycle. The verification plan is updated as and when any changes are made to design inputs.
- Planning includes the scope of the project, tools, test environment, development strategy.

3. Developing

- The test case development identifies a variety of test methods.
- Developing of design inputs which are unambiguous and verifiable.
- Using the output of one test as an input for subsequent tests.
- Traceability links are created to ensure that all the requirements are tested and the design output meets the design inputs.

4. Execution

- The test procedures created in development phase are executed as per the test plan.
- If actual doesn't match with the expected results, such issues are identified and logged as defect at this stage.
- Traceability matrix is created to ensure that all the requirements are tested and the design output meets the design inputs.

5. Reports

- The design verification report gives the detailed summary of verification results which includes the configuration management and test results and defects found during the verification activity.
- Design verification traceability report ensures that all the requirements have been tested and provided with appropriate results.

Syllabus Topic : Monitoring and Control for Design

5.15 Metrics, Monitoring and Control for Design

Design metrics focus on characteristics of program architecture with an emphasis on the architectural structure and the effectiveness of modules or components within the architecture.

Design Metrics

- **Size :** It is defined in terms of:
  - Population: it is the count of operations and classes.
  - Volume: It is the count of population at a given instant of time.
  - Length: It is the measure of a chain of interconnected design elements
  - Functionality: It defines the value delivered to the customer by an OO application.
- **Complexity**: It defines how classes or modules of a design are interrelated.
- **Volatility**: The design changes occur when requirements are modified or when modifications occur in other parts of an application resulting in adaptation of the design component. Volatility component measures the likelihood that a change will occur.
- **Coupling**: It is an indication of the relative interdependence among the modules.
  - The physical connections between the elements of the design like number of collaborations between sub systems, messages passed between objects; all this represent coupling within a system.
  - Coupling allows simple connectivity among modules which result in software that is easier to understand and less prone to errors that occur at one location and propagate throughout a system.
- **Cohesion** : It is the indication of the relative functional strength of a module.
  - A cohesive module performs a single task requiring little interaction with other components in other parts of a program.
  - Cohesion is doing only one thing.

Card and Glass define three software design measures in terms of complexity

1. *Structural complexity* is defined as the number of modules immediately subordinate to a module.
2. *Data complexity* provides an indication of the complexity in the internal interface for a module.
3. *System complexity* is defined as the sum of structural and data complexity.

Monitoring and Controlling Mechanisms

- This task lasts throughout the project and covers the development process
- Monitoring and Controlling is a reporting mechanism that reports about information flows and reviews.
- It controls the volatile changes
- Identifies and controls the risks
- Monitors all key parameters like cost, schedule, risks
- Takes corrective actions when needed

Review Questions

Q. 1 What is the need of system design?

Q. 2 What is the difference between Function Oriented and Object Oriented design.

Q. 3 State the differences between Coupling and Cohesion

Q. 4 What is meant by Abstraction and state its types.

Q. 5 What is the need of problem partitioning?

Q. 6 Draw structure chart and flow chart for 'order processing system'.

Q. 7 What activities are includes in design verification ?

Q. 8 Define the various methods of design verification.

Q. 9 What is the need of monitoring and controlling mechanism ?

Q. 10 State the difference between flowchart and DFD.

□□□