7.4 Maintainability

The application should be maintainable in such a manner that if any new requirement occurs then it should be easily incorporated in an individual module.

7.5 Portability

The application should be portable on any windows based system.

8. Preliminary Schedule

This product must be completed within 7 months.

Review Questions

1. *Explain various steps involved in SRS with suitable example* **MU : Dec.-09, Marks 10**
2. *Write a note on- Software requirements specification.* **MU : May-10, Marks 5**

5.5 Building Analysis Model

Requirement analysis is an intermediate phase between **system engineering** and **software design.**

Requirement analysis produces a software specification.

How is requirement analysis helpful ?

Analyst - The requirement analysis helps the 'analyst' to refine software allocation. Using requirement analysis various models such as **data model, functional model** and **behavioral model** can be defined.

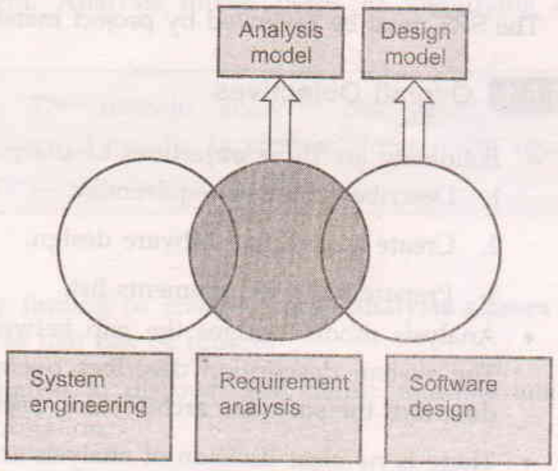

Fig. 5.5.1 Requirement analysis : An intermediate step

Designer - After requirement analysis, the designer can design for data, architectural interface and component level designs.

Developer - Using requirements specification and design the software can be developed.

What are requirement analysis efforts ?

1. Problem recognition

The requirement analysis is done for understanding the need of the system. The scope of the software in context of a system must be understood.

2. Evaluation and synthesis

Following are the tasks that must be done in evaluation and synthesis phase.

i) Define all externally observable data objects evaluate data flow.

ii) Define software functions.

iii) Understand the behaviour of the system.

iv) Establish system interface characteristics.

v) Uncover the design constraints.

3. Modelling

After evaluation and synthesis, using data, functional and behavioral domains the data model, functional model and behavioral model can be built.

4. Specification

The requirement specification (SRS) must be built.

5. Review

The SRS must be reviewed by project manager and must be refined.

5.5.1 Overall Objectives

- Following are **three objectives** of analysis model -
  1. Describe customer requirement.
  2. Create a basis for software design.
  3. Prepare valid requirements list.
- Analysis model bridges the gap between **system description and design model.** The system description describes overall **system functionality** and design model describes the **software architecture,** user interface and component level structure.
- There is no clear division of analysis and design tasks. Some design can be carried out during analysis and some analysis might be conducted during the design of the software.

5.5.2 Analysis Rules of Thumb

Arlow and Neustadt suggested some rules that should be followed during the creation of analysis model. These rules are called **analysis rules of thumb.** They are as follows -

- The analysis model should focus only on requirements that are visible within the business domain. That means there is no need to focus on detailed requirements.

- Each element of analysis model should help in building overall understanding of software. These elements should provide insight into information domain, function and behaviour of the system.
- Minimize coupling within the system. Coupling is basically used to represent the relationship between the two objects by means of functionalities.
- Delay use of infrastructure and other non functional models until detailed software design is obtained.
- Some analysis models should provide the values to the stakeholders. For example, the business stakeholder should focus on the functionalities that are basic demands of the market; software designer must focus on basic design of the product.
- The analysis model should be very simple so that it could be understood easily.

5.5.3 Domain Analysis

It is observed that there are varieties of **patterns** that occur in many applications with in the same business domain. If these patterns are categorized systematically then that allows software engineer to **reuse** them. Analysis model helps in identifying such patterns.

Definition of domain analysis - The domain analysis can be defined as identification of common requirements from a specific application domain, for reusing them on multiple projects within that specific application domain.

How to do domain analysis ?

The domain analysis can be done by finding or creating those **analysis classes** that are having common functions and features that can be **reused.**

It is the responsibility of domain analyst to discover and define reusable analysis patterns, analysis classes and related information.

What are the advantages of domain analysis ?

1. The domain analysis done for finding the reusable design patterns and executable software components helps in building the system or product very efficiently.
2. It reduces the cost of development of design patterns.

Review Question

1. *Explain the importance of modeling practices. Explain the principles of analysis modeling.*

5.6 Elements of Analysis Model

Analysis modelling approach	
Structured approach	**Object oriented approach**
The analysis is made on data and processes in which data is transformed as separate entities.	The analysis is made on the classes and interaction among them in order to meet the customer requirements.
Data objects are modelled in such a way that data attributes and their relationship is defined in structured approach	Unified Modelling Language (UML) and unified processes are used in object oriented modelling approach.

But the commonly used analysis model combines features of both these approaches because the best suitable analysis model bridges the software requirements and software design.

Following are the elements of analysis model -

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

Following Fig. 5.6.1 illustrates the elements of analysis model.

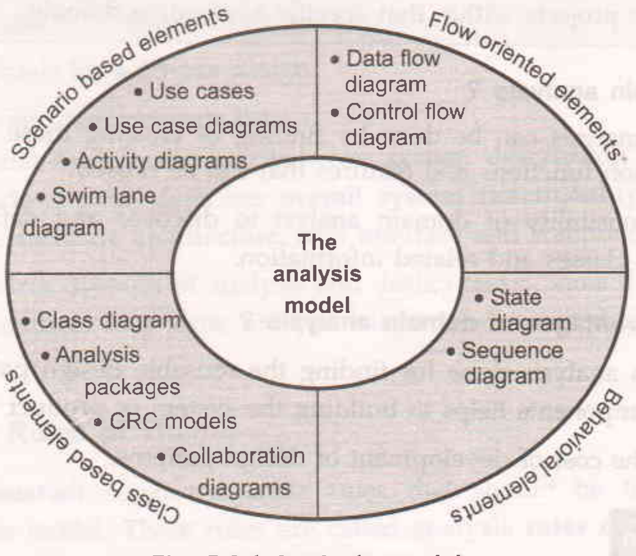

Fig. 5.6.1 Analysis model

Review Question

1. *Explain the analysis model elements with models created in each.*

6.1 Introduction

Software design is model of software which translates the requirements into finished software product in which the details about software data structures, architecture, interfaces and components that are necessary to implement the system are given.

6.1.1 Designing within the Context of Software Engineering

- Software design is at the core of software engineering and it is applied irrespective of any process model.
- After analysing and modelling the requirements, software design is done which serves as the basis for code generation and testing.
- Software designing is a process of translating analysis model into the design model.
- The analysis model is manifested by scenario based, class based, flow oriented and behavioural elements and feed the design task.
- The classes and relationships defined by CRC index cards and other useful class based elements provide the basis for the **data or class design.**
- The **architectural design** defines the relationship between major structural elements of the software. The architectural styles and design patterns can be used to achieve the requirements defined for the system.

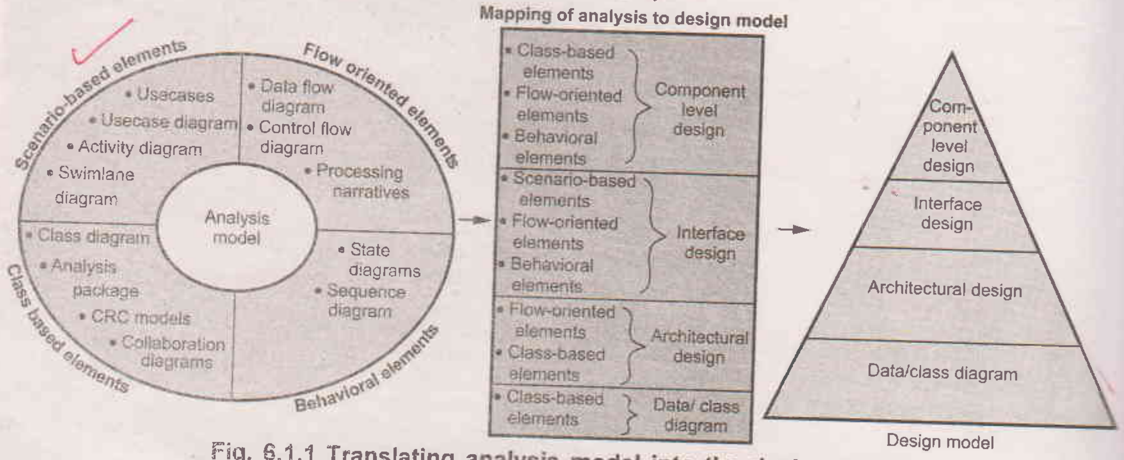

Fig. 6.1.1 Translating analysis model into the design model

- The **interface design** describes how software communicates with systems. These systems are interacting with each other as well as with the humans who operate them. Thus interface design represents the flow of information and specific type of behaviour. The usage scenarios and behavioural models of analysis modelling provide the information needed by the interface design.

- The component-level design transforms structural elements of software architecture into procedural description of software module. The information used by the component design is obtained from class based model, flow based model and behavioural model.
- Software design is **important** to assess the **quality** of software. Because design is the only way that we can accurately translate the user requirements into the finished software product.
- Without design unstable system may get developed. Even if some small changes are made then those changes will go fail. It will become difficult to test the product. The quality of the software product can not be assessed until late in the software process.

6.2 Design Process

Software design is an iterative process in which the requirements are translated into the **blueprint** of the software. Initially the software is represented at high level of abstraction, but during the subsequent iterations data, functional and behavioural requirements are traced in more detail. Thus refinement made during each iteration leads to design representations at much lower level of abstraction. Throughout the software design process the quality of the software is assessed by considering certain characteristics of the software design.

Characteristics of Good Design

1. The good design should implement all the requirements that are explicitly mentioned in the analysis model. It should accommodate all the implicit requirements demanded by the customer.
2. The design should be simple enough so that the code developer, code tester as well as those who are supporting the software will find it readable and understandable.
3. The design should be comprehensive. That means it should provide a complete picture of software, addressing the data, functional and behavioural domains from an implementation perspective.

6.3 Design Principles

MU : May-11, Marks 10

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.

Review Question

1. *List down and explain the software design principles.* **MU : May-11, Marks 10**

6.4 Design Concepts

MU : Dec.-09,10,11,12, May-09,10,12,13, Marks 10

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

Following are certain issues that are considered while designing the software -

- Abstraction
- Modularity
- Architecture
- Refinement
- Pattern
- Information hiding
- Functional independence
- Refactoring
- Design classes

6.4.1 Abstraction

The abstraction means an ability to cope up with the complexity. Software design occurs at different levels of abstraction. 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.

6.4.2 Modularity

- The software is divided into separately named and addressable components that called as **modules.**
- Monolithic software is hard to grasp for the software engineer, hence it has now become a trend to divide the software into number of products. But there is a co-relation between the number of modules and overall cost of the software product. Following argument supports this idea -

"Suppose there are two problems A and B with varying complexity. If the complexity of problem A is greater than the complexity of the problem B then obviously the efforts required for solving the problem A is greater than that of problem B. That also means the time required by the problem A to get solved is more than that of problem B."

The overall complexity of two problems when they are combined is greater than the sum of complexity of the problems when considered individually. This leads to **divide and conquer strategy** (according to divide and conquer strategy the problem is divided into smaller subproblems and then the solution to these subproblems is obtained) . Thus dividing the software problem into manageable number of pieces leads to the concept of modularity. It is possible to conclude that if we subdivide the software indefinitely then effort required to develop each software component will become very small. But this

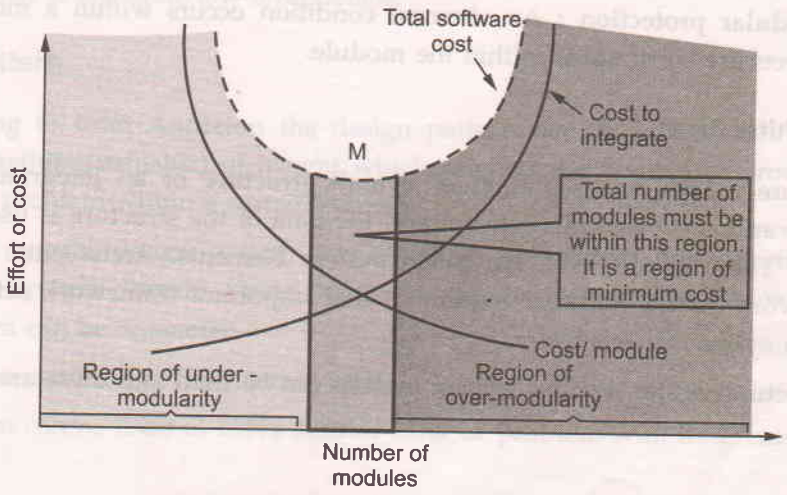

Fig. 6.4.1 Modularity and software cost

conclusion is invalid because the total number of modules get increased the efforts required for developing each module also gets increased. That means the cost associated with each effort gets increased. The effort (cost) required for integration these modules will also get increased. The total cost required to develop such software product is shown by following Fig. 6.4.1.

The above Fig. 6.4.1 provides useful guideline for the modularity and that is - **overmodularity or the undermodularity while developing the software product must be avoided.** We should modularize the software but the modularity must remain near the region denoted by **M**

- Modularization should be such that the development can be planned easily, software increments can be defined and delivered, changes can be more easily accommodated and long term maintenance can be carried out effectively.
- **Meyer** defines five criteria that enable us to evaluate a design method with respect to its ability to define an effective modular system :
  1. **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.
  2. **Modular composability :** A design method enables existing design components to be assembled into a new system.
  3. **Modular understandability :** A module can be understood as a standalone unit. It will be easier to build and easier to change.
  4. **Modular continuity :** Small changes to the system requirements result in changes to individual modules, rather than system-wide changes.
  5. **Modular protection :** An aberrant condition occurs within a module and its effects are constrained within the module.

6.4.3 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.

6.4.4 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.

6.4.5 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.

6.4.6 Information Hiding

Information hiding is one of the important property of effective modular design. The term information hiding means the modules are designed in such a way that information contained in one module cannot be accessible to the other module (the module which does not require this information). Due to information hiding only limited amount of information can be passed to other module or to any local data structure used by other module.

The **advantage** of information hiding is basically in testing and maintenance. Due to information hiding some data and procedures of one module can be hidden from another module. This ultimately **avoids** introduction of **errors** module from one module to another. Similarly one can make **changes** in the desired module without affecting the other module.

6.4.7 Functional Independence

- The functional independence can be achieved by developing the functional modules with single-minded approach.
- By using functional independence functions may be compartmentalized and interfaces are simplified.
- Independent modules are easier to maintain with reduced error propagation.
- Functional independence is a key to good design and design is the key to software quality.
- The major benefit of functional independence is in achieving effective modularity.
- The functional independence is assessed using two qualitative criteria - **Cohesion** and **coupling.**

6.4.7.1 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.

6.4.7.2 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.

Sr. No.	Coupling	Cohesion
1.	Coupling represents how the **modules are connected** with other modules or with the outside world.	In cohesion, the cohesive module performs **only one thing.**
2.	With coupling **interface complexity** is decided.	With cohesion, **data hiding** can be done.

3.	The goal of coupling is to achieve **lowest coupling.**	The goal of cohesion is to achieve **high cohesion.**
4.	Various types of couplings are - Data coupling, Control coupling, Common coupling and Content coupling.	Various types of cohesion are - Coincidental cohesion, Logical cohesion, Temporal cohesion, Procedural cohesion and Communicational cohesion.

6.4.8 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.

Review Questions

1. *Write a note on - Design concepts and principles.* **MU : Dec.-09, May-12, Marks 10**
2. *Explain the fundamental software design concepts.* **MU : May-10, 13, Marks 10**
3. *What is coupling and cohesion ? What should good design contain ?* **MU : May-09, Marks 10**
4. *Compare and contrast coupling and cohesion.* **MU : Dec.-09,12, May-12, Marks 10**
5. *Discuss and compare coupling and cohesion in brief.* **MU : May-10, Marks 10, Dec.-11, Marks 5**
6. *Explain cohesion, coupling and purpose of modular design.* **MU : Dec.-10, Marks 10**

6.5 Design Model

- The **process dimension** denotes that the design model evolutes due to various software tasks that get executed as the part of software process.
- The **abstract dimension** represents level of details as each element of analysis model is transformed into design equivalent.
- In following Fig. 6.5.1 the **dashed line** shows the boundary between analysis and design model.

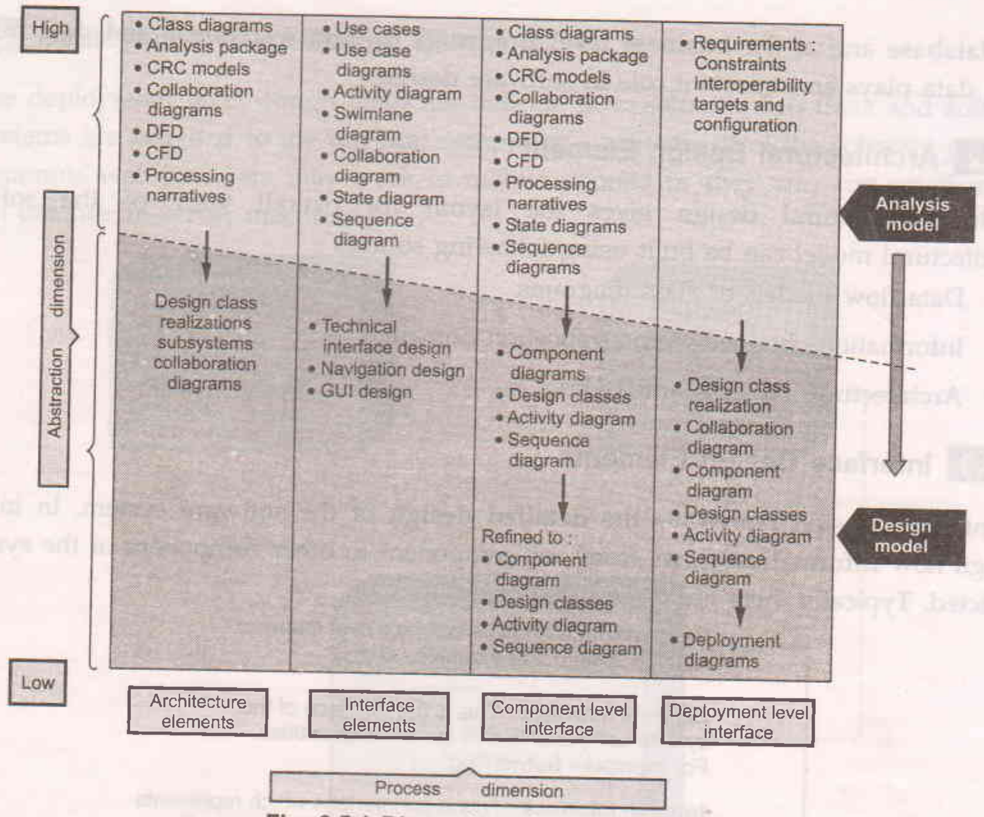

Fig. 6.5.1 Dimension of design model

- In both the analysis and design models the same UML diagrams are used but in analysis model the UML diagrams are abstract and in design model these diagrams are refined and elaborated. Moreover in design model the implementation specific details are provided.
- Along the horizontal axis various elements such as architecture element, interface element, component level elements and deployment level elements are given. It is not necessary that these elements have to be developed in sequential manner. First of all the preliminary architecture design occurs then interface design and component level design occur in parallel. The deployment level design ends up after the completions of complete design model.

6.5.1 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