## UNIT I

### Chapter 1 : Introduction 1-1 to 1-26

**Syllabus :** Introduction to software design, design methods - procedural / structural and object oriented, Requirement Vs Analysis Vs Architecture Vs Design Vs Development, 4+1 Architecture, case study of transferring requirement to design, UP, COMET use case based software life cycle, Introduction to UML - Basic building blocks, Reusability, Use case modeling, Use case template.

**Case study :** Transferring requirements into design using advanced tool.



## UNIT II

### Chapter 2 : Static Modelling 2-1 to 2-22

**Syllabus :** Analysis Vs Design, Class diagram - Analysis - Object and classes finding analysis and Design - design classes, refining analysis relationships, Inheritance and polymorphism, Object diagram, Component diagram - Interfaces and components, deployment diagram, Package diagram.

**UNIT III**



**Syllabus :** Introduction and Interaction overview diagram, sequence diagram, Timing diagram, Communication diagram, Advanced state machine diagram, Activity diagram.



**UNIT IV**



**Syllabus :** Introduction to Architectural design, overview of software architecture, Object oriented software architecture, Client server Architecture, Service oriented Architecture, Component based Architecture, Real time software Architecture.



**UNIT V**



**Syllabus :** Introduction to Creational design pattern – singleton, Factory ,Structural design pattern- Proxy design pattern, Adapter design pattern, Behavioral – Iterator design pattern, Observer design pattern.

**UNIT VI**



**Syllabus** : Introduction to testing, Error, Faults, Failures, verification and validation, White Box Testing, Black Box Testing, Unit testing, Integration testing, GUI testing, User acceptance Validation testing, integration testing, scenario testing, performance testing. Test cases and test plan. Case studies expected for developing usability test plans and test cases.



□□□

UNIT I

CHAPTER 1

# Introduction

**Syllabus Topics**

Introduction to software design, design methods - procedural / structural and object oriented, Requirement Vs Analysis Vs Architecture Vs Design Vs Development, 4+1 Architecture, case study of transferring requirement to design, UP, COMET use case based software life cycle, Introduction to UML - Basic building blocks, Reusability, Use case modeling, Use case template **Case study** – Transferring requirements into design using advanced tool.

**Syllabus Topic : Introduction to Software Design**

## 1.1 Introduction to Software Design

### 1.1.1 Basic Design Concepts

**Q.** What is the role of designing in any system or project implementation? Explain model based software design and development. **(6 Marks)**

☞ **Role of designing**

At very early stage of human evolution, our ancestors learnt and realized the importance of information gathering and representation. They found many ways to preserve information that can be used by their upcoming generations, for example drawing the paintings, forming sculptures. Later many civilizations with skilled man power started making small scale plans or prototypes for their large projects. They used to gather all the required information and start designing and modelling it in architecture.

☞ **Model based software design and development**

The small prototype (model or design) of the actual plan provides a greater scope of refinement in design and also provides greater scope for improvement in existing design. It also provides detailed understanding and illustration of project to be built. With this philosophy, modern information systems have no exception. Software Systems are one of the most important modern information systems. Success of Software Systems largely depends on how carefully it has been designed and crafted at very early stage of its inception. Design process however in software projects are very well defined and need significant effort before starting the actual implementation or coding. The approach of design is often termed as "Model based Software Design and Development".

Models are formed based on well defined specification and referred at all subsequent stage of system implementation. Different design methods may result in different models for software systems. Different models may have different appearance to the user and may cover different aspect of system implementation. There are modeling languages that specify how the model itself should get implemented. UML (Unified Modeling Language) is a graphical modeling language that helps us to design the models that covers different views of the system. We will study UML and its specification in subsequent section. For now we will try to understand and elaborate the software design architecture.

### 1.1.2 Software Design and Major Classification

In previous section, we have understood the role of design and modeling in any kind of system development. In case of modern information system, like a software system, the design and modeling plays so important role that there is need of a separate dedicated branch of study in Computer Science which focuses on design methodologies, software architecture and design patterns. Design of software systems cannot be discussed in isolation with software architecture.

Software architecture deals with entire system architecture in form of components and their interconnection. Designer often starts keeping in mind that system architecture should be clearly visible to all the stake holders of the system at early stage. Nicely designed system architecture will enable developer team to start with the implementation with confidence.

Testing team can refer the architecture and start with early preparation of test cases. Technical representative of client can have a deeper view of the system at early stage and can make sure that client is investing for a valued product indeed. So all the stakeholders like developers, clients, end-users of the product, project manager and test engineers get benefitted by a good architecture. We will cover few more aspects of design and architecture of software system in this section.

### Software design

**Q. What is software design? (4 Marks)**

- A software design is the process of implementing the software. The design process has very significant role in entire software development. A well defined design process results in a good design and a good design results in a successful product.
- Software design is often considered as initial steps towards the solution with available capabilities. There could be multiple designs for the same software system covering different aspects of the system to be developed.
- The term **Software design** is used to refer both, the **process** of designing and the design product or the **model**.
- The design process specifies the sequence of steps with which the architect of software is proceeding towards the design solution. Simply speaking, the well defined steps to form the model.
- A model is nothing but the architecture that describes all aspects of the system that we want to build.
- Design process is where the creativity lies. An experienced and skilled architect needs to be creative also in the modern days of competitive software development strategies.
- An innovative design model makes the coders work more enjoyable and encourages them to apply further innovation in coding strategies. Finally the end user experiences, the difference in their product and the entire contribution puts in overall developing organization at a stage of competitive advantage.

### Classification of software design based on abstraction level

**Q. Describe classification of software design based on abstraction level. (4 Marks)**

- The outcome of software design is often classified in three levels.

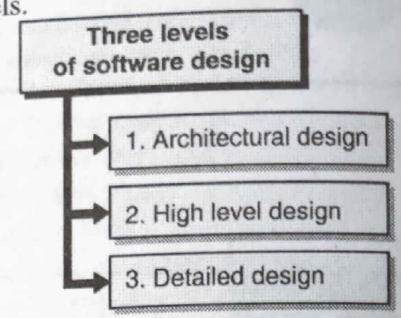


**Fig. C1.1 : Levels of software design**

#### 1. Architectural Design

- The highest level of abstraction of the software system can be represented by architectural design. We can have a very high level view of the entire system and can observe it as many components interacting with each other.
- A simplified architectural of general web application can include user interface or view component, processing component or controller and the data base component that interacts with business logic (algorithms) component model. Fig. 1.1.1 shows the high level view of a web application.

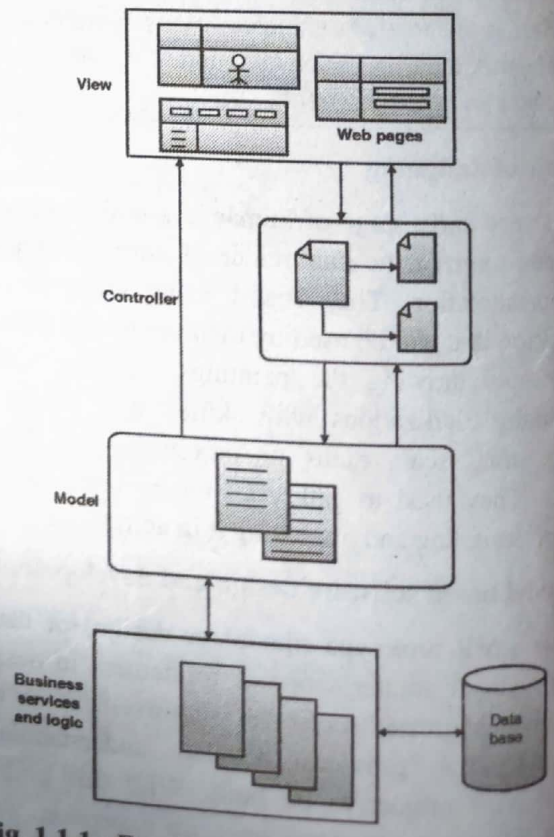


**Fig. 1.1.1 : Possible architecture of modern web application**

➜ **2. High Level Design**

- Breaking the architectural design at component level yields the high level design.
- Here, the concentration is on subsystem design and modules instead of entire system as one. Interaction of the components and interfaces are designed at this level.
- This level still represents the abstract view of the system but at module level.
- Here for example we can have an abstract deign of the sub-system that process the business data and where the business logic is written.
- In most modern web application this component is so large and complex, that high level design of this component takes us back to the architectural level.

➜ **3. Detailed Design**

- Detailed designing is the boundary where we actually switch from deigning to implementation. Each and every module structure is implemented here.
- The interface to the module is identified. For example, in object oriented designing and programming, main components of the system can be identified as packages, classes, objects and interfaces. Their behaviour and properties are encapsulated as methods and variables.

☞ **Fundamentals of design concepts**

**Q. Explain the fundamental design concepts applied in modern software design. (6 Marks)**

Now we will list few **fundamental design concepts** that have been evolved over the years of experience in software designing.

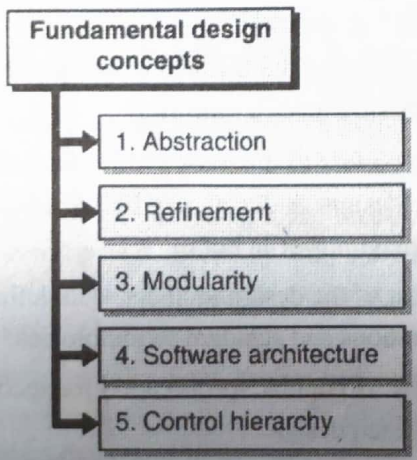


**Fig. C1.2 : Fundamentals of design concepts**

➜ **1. Abstraction**

- The first stage of designing is focused on generalizing our concept of system.
- We design a basic system by picking up the most relevant information and hiding the complexities of the system.

➜ **2. Refinement**

- This is another design aspect which comes into picture in later stage of design process. Here we concentrate on elaboration. Refinement starts once our high level design is ready.
- Then we keep on applying refinement on our design and finally reach the low level programming constructs like packages, Interfaces, Classes, objects, methods variables, etc.

➜ **3. Modularity**

- Complex design can be divided into modules. While designing, it is made sure that module
- A can be treated in isolation if needed. It means that the inputs, outputs or interaction should be well defined for the Module A and if needed, a team T1 can be assigned to work on that module in isolation.
- The assigned development team T1 may not actually have the access to the larger perspective of the system or it may not have access to working Module X with which Module A interacts.

➜ **4. Software Architecture**

- One of the most important design concepts is the overall architecture or the **software architecture** of the system which should be very carefully designed.
- This architecture plays important role in system implementation and kept on getting referred throughout the life time of software system.
- All the stakeholders will always wish to have comprehensive, appealing, referable and unfeigned software architecture at first place.
- A good software architecture always signals a good return of investment for the stakeholders.

➜ **5. Control Hierarchy**

- Another important aspect of system is flow of control and data. Design at some level should also be able to project this aspect of the system.
- Stages of user interaction and then subsystem interactions can be used to show the control hierarchy at high level. At more detailed level control structures

of programs and their calling sequences represents the control hierarchy.

- The above mentioned terms of design are general design concepts. Software architects and expert engineers can add more terms and parameters that they have learnt and experienced.
- Design engineering and process can never be frozen. IT has witnessed few major changes in software systems in past few years.
- Mobile applications and Internet forced us to rethink on traditional design strategies.
- With modern trends and need in software development, we should not hesitate to find, verify and adopt the innovative strategies in software design.

### 1.1.3 Software Architectural Design

- A high level design that focuses on overall structure of the software system is often termed as Software Architecture.
- Software architecture depicts a modular design of the system in the form of components and their interconnection.
- Software architecture also refers the detailed internal design of each module and their interfaces.
- An architecture design for software system that focuses on components and their interconnection is called "Programming-in-the-large".
- An architectural design that focuses on internal design of each component is called "Programming-in-the-small.
- Architectural design covers different aspects of the system at different level of details. One overall architecture depicts entire system as a whole.
- Another can decomposes the system into subsystems and shows the interaction. Also there could be more architecture to show the modular decomposition of each sub-system and so on.
- The design principles and specifications are applied throughout the system architecture design process. At each level there are well defined process and notations to implement the design that is universally accepted.
- These processes, design methods and notations can be combined to implement the creativity and innovation in Software architectural design and results in a successful product.
- One of the most important factors of software system is its quality attributes like performance, maintainability, and security.
- These quality attributes falls under the category of non-functional requirements.
- A good architectural design always tries to convince the viewer that system will satisfy these requirements.

### 1.1.4 Design Notations and Methods

**Q.** Explain few software design methods and notations. **(6 Marks)**

In this section, we explore few important terminologies related to software designing. A short description is provided below. We will keep on referring and elaborating these terminologies in this book.

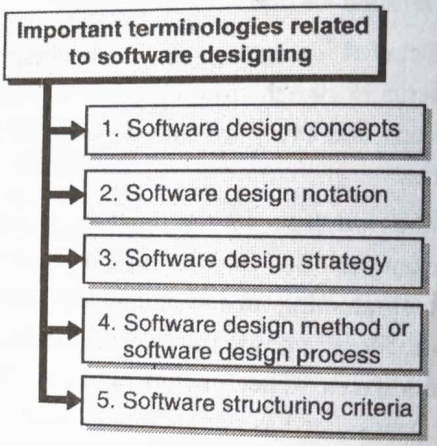


**Fig. C1.3 : Terminologies related to software designing**

➜ **1. Software design concepts**

- This refers to the fundamental design concepts around which we start building our design. The design concept depends heavily on nature and requirement of system.
- **Example :** modern object oriented systems starts with abstraction and information hiding and highlighting the features which are important.

➜ **2. Software design notation**

- This refers to the tools and medium for specifying the design. A design can be specified in words or in diagrams or in both.
- Most of the design includes well defined graphical notations and standard terminologies.
- UML is popular specification for specifying object oriented design.

- The syntax is pre-defined but it is up to designers and architects to represents that syntax in meaningful way.

➜ **3. Software design strategy**

- The overall approach of design is software design strategy.
- **Example :** Procedural, structured or object-oriented are few examples.

➜ **4. Software design method or software design process**

- After the requirement analysis is done and problem definition is completed, software design process starts.
- This is well defined specification to create a design through sequential steps.
- However many times the design process overlaps with requirement analysis where a designing method is applied on gathering and specifying the requirement.
- Major design decisions are made in this phase.

➜ **5. Software structuring criteria**

- This the road map for organizing the software components, defining the interaction and interfaces between them and forming the entire system.
- **Example :** One example is object structuring criteria where all the objects are identified and structured together to form the system.

**Syllabus Topic : Design Methods**

## 1.2 Design Methods

### 1.2.1 Evolution of Design Methods

- In earlier days of software development, engineers used to follow very straight forward way of code development. As a prior preparation, they used to develop flowcharts and algorithm. Writing algorithms and flowcharts were the actual design work.
- Later they started adopting the modular way of program development. This way, they have realized the strength of code distribution, sharing and project management.
- One common practice back at that time was to divide the entire system into modules, modules are assigned to separate team that could be implemented in isolation.
- Programming languages like C in early 70's, has started supporting **structured** way of program development. Subroutines and functions of any structured programming language can be used to develop such modules.
- This structured way of system designing was often termed as top down approach or step wise refinement. By the year 1970, this approach has gained popularity among software development organizations.
- By early 1980's a new way of thinking towards the solution has emerged. This was the time when object-oriented analysis and designing starts taking its ground.
- We will keep on discussing this methodology throughout this book. For now we will explore few more aspects conventional designing methodologies.

**Syllabus Topic : Procedural and Structural Design Methods**

### 1.2.2 Procedural and Structural Design Methods

**Q. What are structured and procedural design methods ? (6 Marks)**

- As discussed in previous section, in early 1970's structured programming and designing starts getting popularity among software developers. This kind of designing methodology was based data-flow oriented designing.

☞ **Structured design**

- The objective of structured designing was to organize the entire system in well defined modules. The structured designing was one of the first well documented designing methodologies.
- One important aspect of structured designing is to capture the data flow in the system. The documentation includes well specified construct to define data flows, called the DFDs or data flow diagrams. At later stage of designing, DFDs are mapped into structure charts.
- Two important approaches of designing were introduced with structured designing. They were called coupling and cohesion, two important aspects of modularity.
- Structured analysis is based on DFD's designed during early structured design phase.

☞ **Procedural design**

- Procedural design is even more conventional design methodology where design decisions are made even when the system functionality is not understood properly or even when all the information is not available.

- Particular ordering of decision making process is followed in procedural designing and it down not get affected by the nature of system.
- Procedural designing is highly risky in modern software system where designing strategies with changing requirements even during the phase of development.
- Structured designing has also limitation with many modern days application development where systems have very dynamic behaviour. Now we will switch to more familiar, popular and robust strategy called the object oriented designing methodology.

## Syllabus Topic : Object Oriented Analysis and Design Methods

### 1.2.3 Object Oriented Analysis and Design Methods

**Q.** Explain the concept of object oriented analysis and designing and its role in modern application software development. **(6 Marks)**

☞ **Concept of object oriented analysis and designing**

The concept "Object Orientation" is not new but the designing methodology starts taking its place late 1980s. This was the time when developers started understanding the potential of objects oriented programming and was looking some methodology that can be mapped with programming constructs available in object oriented language. There was need of designing methodology where the concepts and principles of objects orientation can be applied in designing and later same could be applied in coding. With a well specified, widely accepted object oriented analysis and designing methods in place, architects gets the confidence that their design will be supported by an object oriented programming language. Developers can understand the design very well as they can relate designing constructs with programming constructs they would be using. A smooth and quick transition from designing to coding is possible with object orientation. So overall a software organization gets a common platform for system architects and developers.

☞ **Role of object oriented analysis in modern application**

- Object orientation is way of thinking a problem with real world perspective. The concept of the system in mind is mapped in the objects, their properties behaviour and interaction.
- Object orientation avoids the problems and pitfalls associated with conventional designing methodology.
- The concepts that we mostly address in object orientation approach is classes, objects, inheritance, generalization, specialization, polymorphism and abstraction.
- **Objects** are the central concept in object orientation. It represents the abstraction of any real object in problem domain. It reflects the capability of the system. It reflects what information system will keep and also it reflects what methods it offers to interact with the system.
- Information hiding or abstraction is another central approach of object oriented designing. If a module is well contained with its own information, it will be less dependent on other modules and hence changes can be easily done without affecting the other modules.
- **Object orientation analysis** is the first method applied in object oriented methodology of software development. This is the analysis phase of software development. The mapping of real world to programming world is performed here. Objects are identified along with their properties and behaviour. The phase is often called **object modeling**.
- One approach of object modeling is based on pure data modeling, the concentration is on information modeling where we do not consider the behaviour or methods of the objects. E-R diagrams are formed and the logical database is designed. Here we focus on attaching more and more relevant information to the identified objects. Identified objects are nothing but data base objects. The objects in data modeling are called **entities.**
- In object oriented modeling, we do not concentrate on pure data modeling, so the objects are identified not only with data but also with operations. The objects are then mapped into software classes. The classes can be used to depict advanced modeling concepts like specialization, generalization, aggregation etc.
- The above mentioned way of identifying the objects and mapping them into classes is called **static object modeling or domain modeling**.
- The static object modeling can be implemented by widely accepted and standardized modeling language

called UML, the unified modeling language. We will overview the UML notation in upcoming section.

**Syllabus Topic : Requirement vs Analysis vs Architecture vs Design vs Development**

## 1.3 Requirement vs Analysis vs Architecture vs Design vs Development

### 1.3.1 Requirement vs Analysis

**Q. Compare requirement and analysis. (4 Marks)**

- These two are most elementary and important phases of software development. Often, in practice, both are combined to be termed as requirement analysis.
- The two phases often termed as requirement gathering and requirement analysis have subtle different.

| Sr. No. | Parameters | Requirement Gathering | Requirement Analysis |
|---|---|---|---|
| 1. | Process Flow | This is the process of collecting requirement for the proposed systems from users, customers and other stakeholders. | This is the process of logical analysis of gathered requirement to build the foundation of system architecture. |
| 2. | Stakeholders (Involvement) | Business analyst can be involved in the task of requirement gathering. | Business or system analyst, developers or architects can be involved in analysis task |
| 3. | Method (Style) | Requirement gathering comparatively more informal and can be gathered and documented in informal manner. | Requirement analysis is a formal and project specific process where well defined analysis document is prepared. |
| 4. | Outcome | The outcome of requirement gathering is an informal document that need to be revised before handing over to the developers and designers. | The outcome of requirement analysis is a formal requirement document that can be handed over to the developers or architects. |

### 1.3.2 Analysis vs Architecture

**Q. Differentiate analysis and architecture. (4 Marks)**

- Analysis is an initial phase of software development lifecycle and builds the foundation for system implementation.
- After the requirement analysis gets completed, the concentration of project development shifts towards architecture building for system.

| Sr. No. | Parameters | Requirement Analysis | Architecture Formation |
|---|---|---|---|
| 1. | Process Flow | This is the process of logical analysis of gathered requirement to build the foundation of system architecture. | This is the early approach of system design. The initial high level system design decisions are made in this phase. |
| 2. | Stakeholders (Involvement) | Business or system analyst, developers or architects can be involved in analysis task. | System architects, analysts or developers are engaged in this phase. |
| 3. | Outcome | The outcome of analysis is formal document often termed as SRS or software requirement specification document. | The outcome of architecture formation phase is high level system design or architectural design document. |
| 4. | Artefact and Elements | The use case scenarios and stories can be stated and formulated is analysis phase. | Individual use case scenarios are explored and refined during the architecture formulation. |

### 1.3.3 Architecture vs Design

**Q.** Compare architecture vs design. **(4 Marks)**

- Architecture formation and system design both are the phases of software development which serves as the foundation for system implementation.
- Architectures refers to the high level design of the system where high level system and business decision are incorporated. Design refers to the detailed design of the individual components of the system.

| Sr. No. | Parameters | Architecture Formation | Detailed Design |
|---|---|---|---|
| 1. | Process Flow | This is the process of logical analysis of gathered requirement to build the foundation of system architecture. | This is the early approach of system design. The initial high level system design decisions are made in this phase. |
| 2. | Stakeholders (Involvement) | System architects, analysts or developers are engaged in this phase | Development team or architects are involved in detailed designing phase. |
| 3. | Outcome | The outcome of architecture formation phase is high level system design or abstract design document. | The outcome of detailed designing phase is low level system design or detailed design document. |
| 4. | Artefact and Elements | Individual use case scenarios are explored and refined during the architecture formulation. | Specifications of the program modules and interfaces are represented in design. Class diagrams, object diagrams etc. are used for the purpose. |

### 1.3.4 Design vs Development

**Q.** Compare design and development. **(6 Marks)**

- Design phase of the software development builds foundation complete projects implementation in the form of programming codes. Implementation refers to program implementation phase of project development.
- The differences between design and development (implementation) is shown in Table 1.3.1.

**Table 1.3.1**

| Sr. No. | Parameters | Design | Development |
|---|---|---|---|
| 1. | Process Flow | This is the early approach of system design. The initial high level system design decisions are made in this phase. | Implementation refers to the programming activity of software development. |
| 2. | Stakeholders (Involvement) | Development team or architects are involved in detailed designing phase. | Programmers and developers and test engineers are involved implementation phase. |
| 3. | Outcome | The outcome of detailed designing phase is low level system design or detailed design document. | The outcome of implementation is a working software product or serviceable entity. |
| 4. | Artefact And Elements | Specifications of the program modules and interfaces are represented in design. Class diagrams, object diagrams etc. are used for the purpose. | Programming constructs, concepts and patterns are implemented in the phase. |

**Syllabus Topic : 4+1 Architecture**

## 1.4 4+1 Architecture

➔ **(Dec. 2016)**

**Q.** What is 4+1 architecture view model? Explain with suitable diagram. **SPPU - Dec. 2016, 5 Marks**

- We previously discussed that the design can have multiple views and so the architecture. These views cover different aspects of the system to be implemented and look at it from different perspectives.
- We can understand it by taking example architecture of new house to be built. We can have four different architectural perspective of same house architecture shown in Fig. 1.4.1.
- Similarly multiple views have been advocated by many system architects for software architecture.

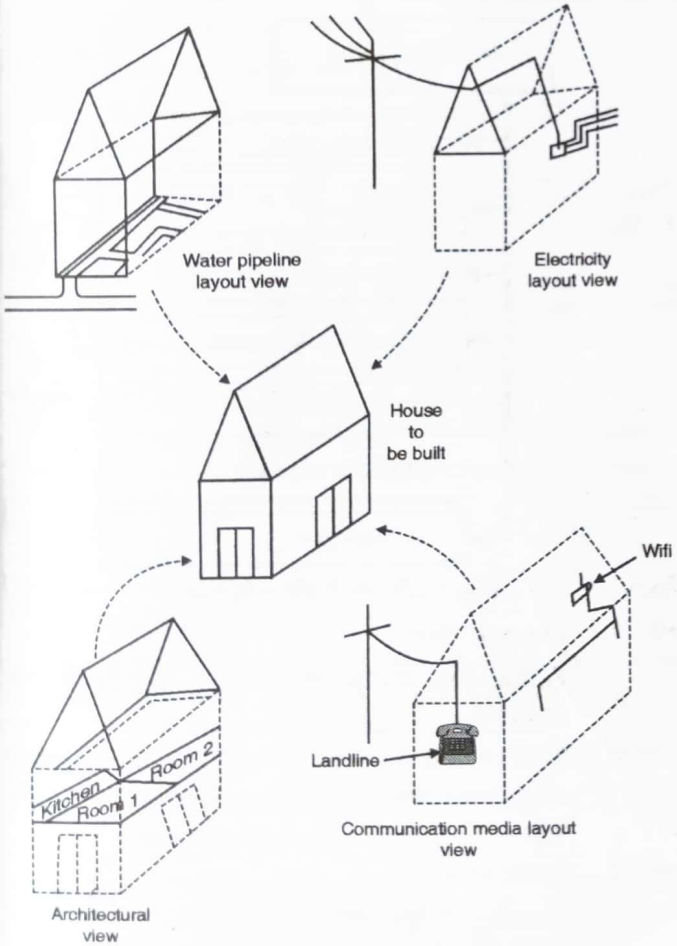


**Fig. 1.4.1: Four different architectural view for house to be built**

- One remarkable is 4+1 view model designed by Phillippe Kruchten based on multiple and concurrent views. Fig. 1.4.2 illustrates the 4+1 architecture view model.

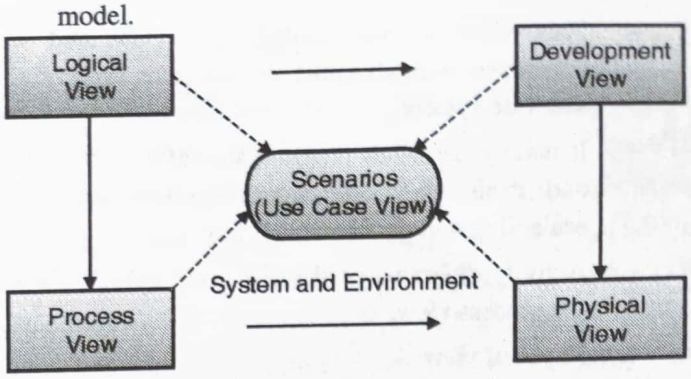


**Fig. 1.4.2 : The 4+1 architecture view model.**

This view is defined from different stakeholder's perspective like end-user, developer, project manager.

☞ **Views of system**

**Q.** Explain different views of software architecture. **(4 Marks)**

- We are listing here four views of the system namely.

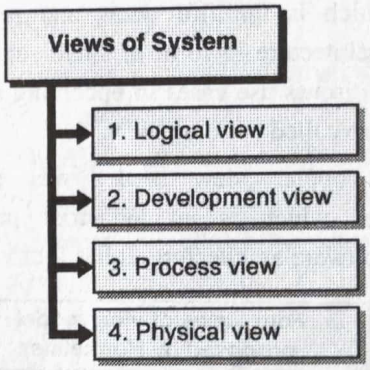


**Fig. C1.4 : Four views of system**

➔ **1. Logical View**

- This view is also called static modeling view and is concerned with the functionalities offered to the end-user of the software system.
- The models in this view can be represented by sequence diagrams, class diagrams and communication diagrams.

➔ **2. Development view**

- This view is also called subsystem and component design view or the implementation.
- This is the developer's view, contributes and project development.
- Component diagrams and Package diagrams defined in UML are often used to implement this view.

➜ **3. Process view**

- This view is also called task view and is concerned with dynamic or run time aspects of software system.
- It mainly describes program structure at run time and deals with concurrency, system integrity, scalability and performance.
- Activity diagrams of the UML are used to form the process view.

➜ **4. Physical view**

- Physical view is concerned with deployment, administration and environment of the system.
- This is often the system engineer's point of view. Supporting software, tools and their interaction with the system is often represented by this view. Deployment diagrams are used in this view.

### 1.4.1 Scenarios

- In 4+1 architectural view model, we have scenario at centre which is the fifth view and represented by system architecture in form of small set of use cases. (We will discuss use cases in upcoming section). This view is also called use case view.
- Another popular view model was proposed by Hofmeister which is an industrial perspective of applied software architecture. It has four views.

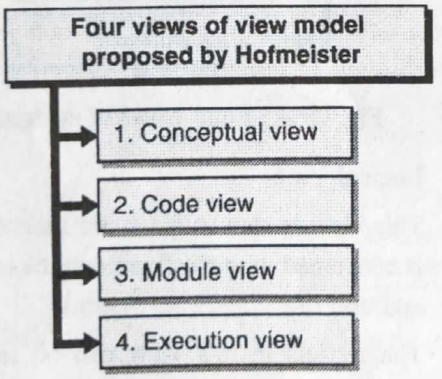


**Fig. C1.5 : Views of view model proposed by Hofmeister**

➜ **1. Conceptual view**

- The primary design elements are described in conceptual view.
- The relationship of the design elements are also described in this view.

➜ **2. Code view**

- Code view is the developers view and describe the structure of source code and their classification into objects.
- It also describes code grouping into libraries and packages.

➜ **3. Module view**

It describes the modules or sub systems and their interactions.

➜ **4. Execution view**

This is somewhat similar to process view and mainly deals with run time, concurrent and distributed execution perspective.

☞ **UML Specified Software Architecture**

Now, we will discuss the different views for software architecture specified by UML and are most important in modern software development. Here, we are not confined to four or five views but as per the requirements of the project design, specific views can be included. We can take it as superset of 4+1 view model and few other view models with standardization of UML.

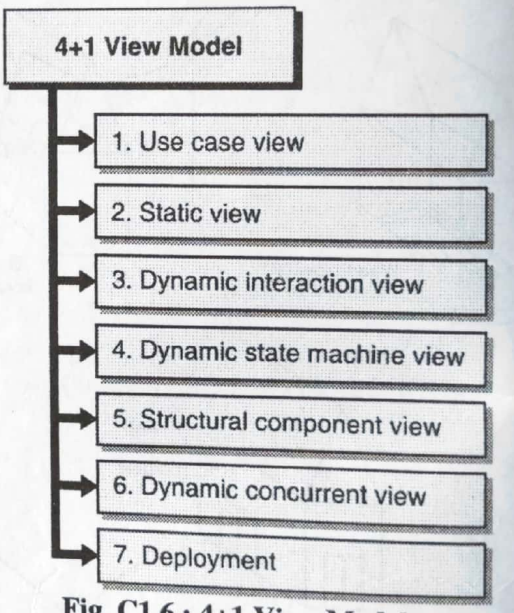


**Fig. C1.6 : 4+1 View Model**

➜ **1. Use case view**

- In this view, interactions of actors and systems are depicted.
- This view is often the first step while designing the system and acts as the functional requirement view.

➜ **2. Static view**

- Classes are the declaration of system properties and interaction. **Static view** is captured by defining Classes and showing the relationship with some standardized terms like aggregation, generalization, specialization etc.

➜ **3. Dynamic interaction view**

- When we are thinking of system structure at run time, then classes takes the form of objects.

- Then we have to consider the method invocation and message passing between the objects.
- The **dynamic interaction** view is used to capture this interaction in the form of objects and their communication. Communication diagrams are often used to show this view.

➜ **4. Dynamic state machine view**

- The actual flow of control in system depicted by the dynamic machine state view.
- State machine is used to show the sequencing of control components. UML's state chart is used for the purpose.

➜ **5. Structural component view**

- This is the component view of the system where the interaction of component, their interfaces and the ports where they have to be connected are shown.
- Class diagrams are often used to show the Structural component view.

➜ **6. Dynamic concurrent view**

- The distributed aspects of the system are captured in the dynamic concurrent view.
- The concurrent components which will be getting executed at remote distributed nodes are depicted in this view.
- Concurrent communication diagrams are used to show such concurrency.

➜ **7. Deployment view**

- The actual environment of software deployment including network components, hardware nodes and supporting software like containers are captured in this view.
- UML's deployment diagrams are used to show this view.

**Syllabus Topic : Case Study of Transferring Requirement to Design**

## 1.5 Case Study of Transferring Requirement to Design

**Note :** Refer Use case modeling elaborated in next section before exploring this case study. Advanced CASE tools can be used to generate use case and class diagrams by students. Student can take assignment to further refine and realize the use case using openly available tools like ArgoUML

☞ **Case Study for Service Oriented Architecture : e-shopping system**

### A. Problem Description

e-shopping (also known as Internet shopping, Online shopping, e-web store etc.) is an example of modern e-commerce system that allows users, buyers or consumers to buy the goods or services online. The system is accessible online through web browser or mobile app. Today, it is taking the form of m-commerce and sometimes deeply integrated with m-commerce where users access the system through mobile optimized sites or apps.

The e-shopping system is web based online system. A buyer can browse the catalog, pick the items in shopping cart, update the shopping cart, check out and make the payment. Customer/buyer can track the placed order and cancel the order. Customer can write the feedback and can submit the product review. Supplier/seller can login in the system can process the order, update the order status, update the catalog to display seasonal offers etc.

Customer provides personal information like address and card (debit /credit) details which are stored in system in users account. Once information is verified by the system, the order gets placed and system takes the necessary action to charge the customer's credit/debit account. An email is sent to the registered customer email id stating successful placement of order. Seller is notified with order details. Seller checks and verifies the order then confirms the order with expected delivery date and courier information. A mail is generated and sent to the customer as a confirmation from seller.

### B. Use case modelling for case study : e-shopping system

- A high level use case is illustrated for the e-shopping system where the primary actor is buyer and seller.
- We can make a formal description table for various scenarios like view and pick items, make order request, process order, etc.
- We can also design activity diagrams for these scenarios with suitable assumptions. For now will be covering the Use case diagram and proceed towards static structuring of system. The use case diagram is shown in Fig. 1.5.1.

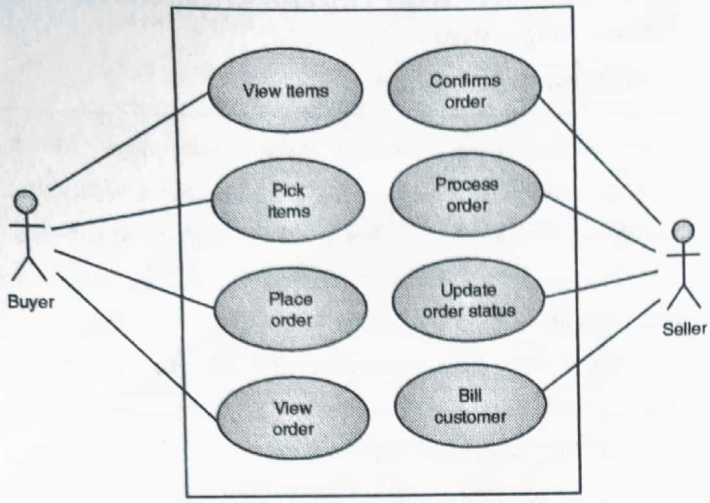


**Fig. 1.5.1 : Use Case : e-shopping system**

## C. Static Modeling : Class and Object structuring

- In static modeling for an application to be formed as service oriented architecture, services are designs high level classes.
- The primary services are identified here with their respective entity classes. Entity classes are represents the entities in application domains which has certain attributes.
- A service class provides access to entity classes. The stereotypes <<entity>> and <<service>> are used to represent and entity and service classes respectively.
- Fig. 1.5.2 shows the service and entity classes for e-shopping system. Note that there could be more services and classes that can be added in this diagram. For the sake of simplicity, we picked the important classes that influence the online application.
- For example, Courier Service class can also be added with <<service>> stereotype which provide access to the Parcel <<entity>> class.
- Similarly Email service can be added that has dedicated task of generating emails.

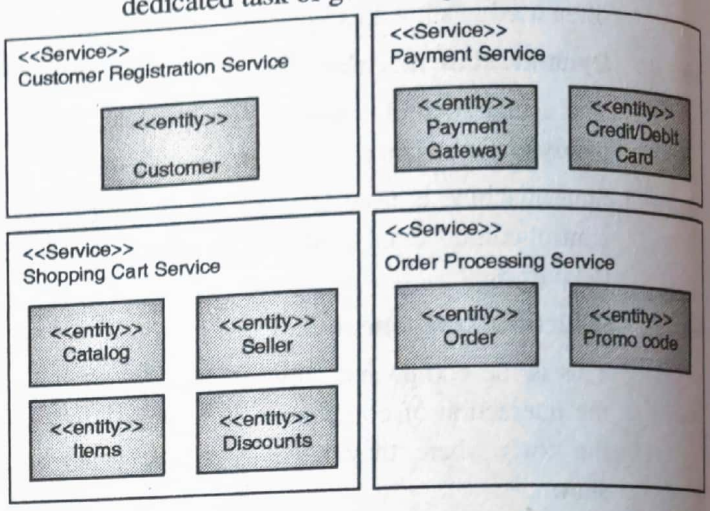


**Fig. 1.5.2 : Service and entity classes for e-shopping system**

## D. Dynamic Modeling for e-shopping system

- System can be dynamically modelled for the different use case scenarios. For example one dynamic modelling is performed to show how the customer interact with system while registering the personal information.
- Another can be shown to capture the *Catalog browsing* or *View Item* use case. Another can be shown for *Order Processing* use case.
- This can be designed with the help of communication diagrams showing messages getting passed between different services.
- One example is dynamic modeling shown in Fig. 1.5.3 that captures the use case where customer/buyer places the order.

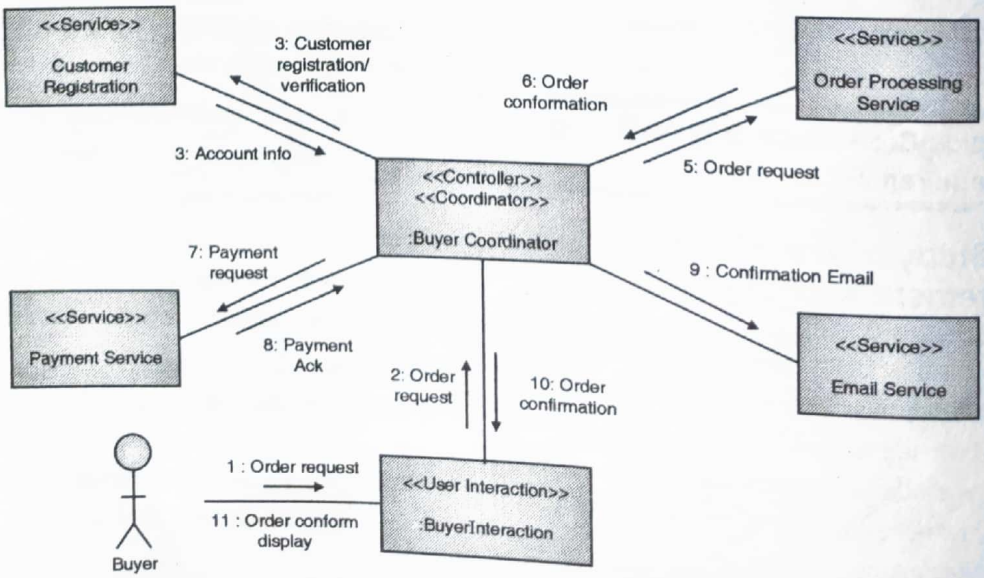


**Fig. 1.5.3 : Communication diagram for Place Order use case**

## E. Service-Oriented software architecture for e-shopping system

- Service oriented architecture for e-shopping system can be organized into three layers: Service layer, Co-ordination layer and User layer.
- Service operations are accessed by one **"provided interface"**. Clients can invoke the appropriate service provided by **interface**.
- Services are represented by individual **components** in the diagram. **Interfaces** are attached to its internal service by well defined **port** which is represented by small square shaped box attached to the **component**.
- The **provided interface** represents a formal contract of the services that consumer gets.
- There could be **"required interfaces"** attached to the service representing the external requirement for the service for its operation. For example **OrderProcessingService** may have one **provided interface (PI)** called **PI_OrderProcessing** and one **required interface (RI)** called **RI_Payment Service.**
- The required and provided interfaces are linked to internal service by required and provided ports respectively.
- Fig. 1.5.4 shows the layered service oriented architecture of e-shopping system. Ports starting with P represent provided ports and ports starting with R represent required ports.

### Role of CASE tool

- Many CASE tools are available for enterprises and individual to transfer requirement into design.
- These computer aided software engineering tools allows conveniently to specify the requirements in the form of stories, uses cases etc.
- These tools allows creation of static as well as dynamic modelling diagrams (both structures and time ordered).

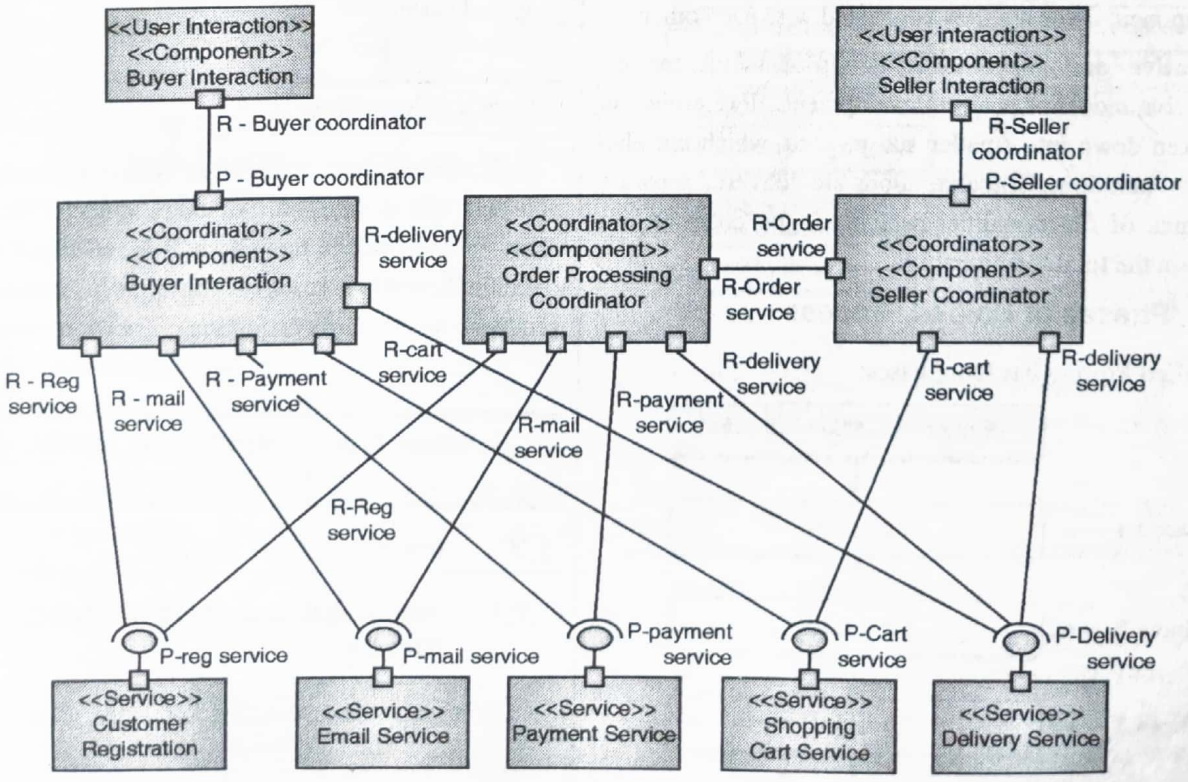


**Fig. 1.5.4 : Service Oriented Architecture for e-shopping system**

**Syllabus Topic : UP**

## 1.6 UP (Unified Process)

Software industry uses some well-defined process model for developing software. This is model is called Software Engineering Process (SEP) or Software Development Process (SDP).

The Unified Software Development Process (USDP) is a popular, widely used industry standard SDP. In short it is also called **Unified Process (UP).**

Unified Process co-exists with UML (Unified Modeling Language) which is a visual language part of the system or project. UP addresses the process part of software development.

There are few widely accepted axioms associated with UP. They are as follows :

1. **Use case and Risk Driven :** This axiom primarily focus on use case scenarios and risk driven management of projects. Risk need to be carefully analyzed, explored and handled in almost all IT projects. UP inherently has capability of addresses the software construction by analyzing risk.
2. **Architecture Centric :** The UP process in centered on robust system architecture. Architecture specifies how system can be divided into different system components how are they connected with each other.
3. **Iterative and incremental :** Up also focuses on iterative model of project development. Here project is broken down into smaller sub projects which are also called iterations. These iterations are delivered separate chunks of functionalities which can be combined to obtain the final functionality.

### 1.6.1 Phases of Unified Process

Unified Process has four phases:

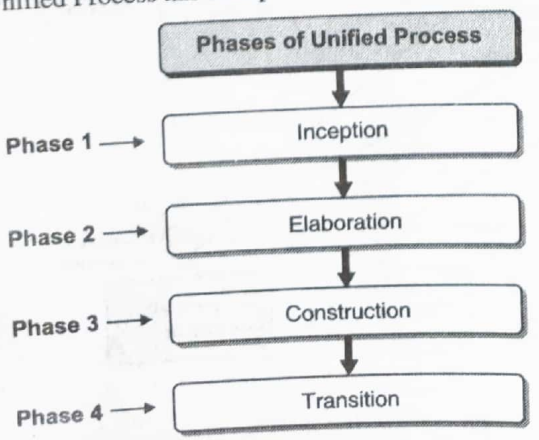


**Fig. 1.6.1: Phases of Unified Process**

**Phase 1 : Inception**

The ground for the project is set in inception phase. The primary focus of inception phase is gathering and analyzing the requirements. Business work flow is understood clearly, feasibility is explored and risk is analyzed in this phase. Deliverables after inception could be project plan, business case, prototypes, vision document etc.

⬇

**Phase 2 : Elaboration**

The deliverables of inception phase is elaborated in this phase. Risk is refined and detailed system construction plan is created. The primary focus in this phase is to create an executable architectural baseline for system construction. Deliverables are the static and dynamic UML model.

⬇

**Phase 3 : Construction**

The architectural baseline created during the elaboration is converted into the final system in construction phase. Construction phase strictly follows the architectural baseline and maintain its integrity. Requirements, analysis and design in finalized in construction phase and implementation is started to build the initial capability of the proposed system.

⬇

**Phase 4 : Transition**

Transition phase starts when system is finally deployed and beta test is finished. All bugs, defects, loopholes are analysed and fixed in this phase. Release of software for its target audience is prepared is this phase. The final software product is ready after completion of transition phase.

**Syllabus Topic : Introduction to UML - Basic Building Blocks**

## 1.7 Introduction to UML

### 1.7.1 Introduction and Basic Building Blocks

**Q.** What is Unified Modeling Language (UML)? List few standard notation and diagrams of UML. **(4 Marks)**

☞ **Basic Building Blocks of UML**

– The user guide of UML specifies that the UML composed of three basic building blocks:

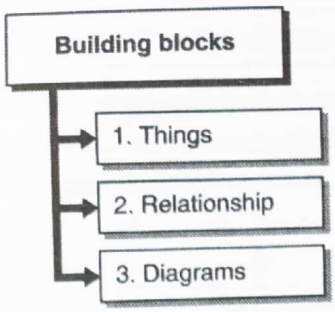


**Fig. C1.7 : Building blocks**

1. Things: The fundamental modeling element.
2. Relationships: The elements specifying the relationship among things
3. Diagrams: The views that provides the pictorial representation of the software system.

➜ **1. Things**

UML things can be categorized as following:

(i) **Structural Things :** The static components of the UML model which are stated as nouns are called structural things. Example: class, interface, collaboration, use case, activity, component etc.

(ii) **Behavioural Things :** The activities between the structural things are represented by behavioural things. They are the verbs of UML model. Example of behavioural things are: interactions, state machines etc.

(iii) **Grouping Things :** The container used to logically group similar kind of elements is called grouping things. Example is packages to group uses cases, classes etc.

(iv) **Annotation Things :** The extra information attached to the diagram elements. These thins captures the ad hoc information and are like sticky notes in diagram.

➜ **2. Relationship**

Relationship specifies how two or more things are connected. The semantic connection between the things are the relationships. The following are the type of relationship:

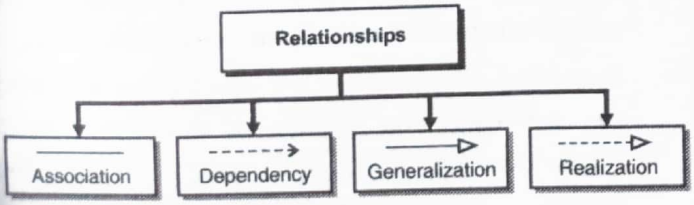


**Fig. 1.7.1**

(i) **Association :** Association is simply the link or the set of links between the objects. This is represented by plane line between the objects.

(ii) **Dependency :** Dependency is special relationship which is used to denote the change in object affecting the state of other object. This is denoted by dotted line and an open arrow.

(iii) **Generalization :** This relationship specifies the connection between general and specific things. It is represented by solid line and closed arrow.

(iv) **Realization** : The relationship between the classifiers where the classifier is implementing the contract for other classifier. This is represented by dotted line and closed arrow.

➜ **3. Diagram**

The views of the UML is called the diagram. There are nine types of diagram in UML. They are shown in the following figure.

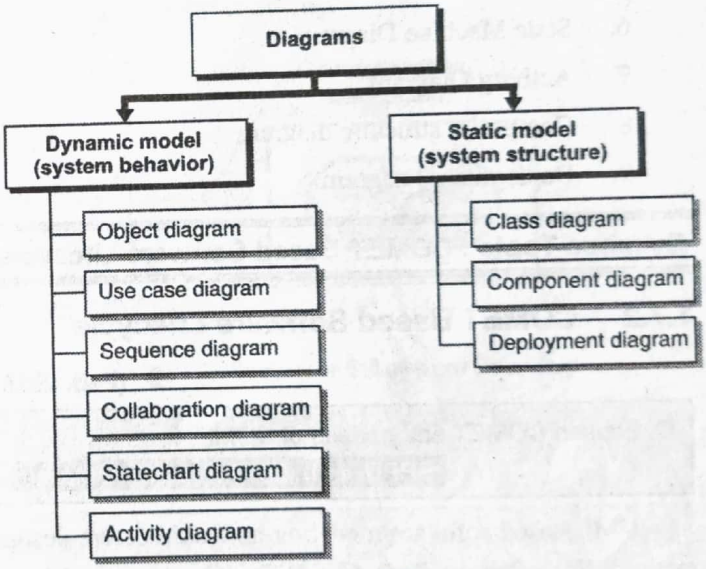


**Fig. 1.7.2**

- UML (Unified modeling Language) is standard modeling language and notation for object oriented analysis and design.
- These notations are used to visualize the design of system and widely accepted among the software development organizations.
- OMG (Object management group) has adopted UML as standard notation for designing in 1997. The most significant contributors in forming the notations and specifications of UML are Grady Booch, Ivar Jacobson and James Rumbaugh. The design and notation of UML was developed by them at Rational Software.

- UML based software modeling and architecture design method is often called COMET (*C*oncurrent *O*bject *M*odeling and Architecture Design M*e*thod). This is specification for single system, and covers the entire project development life cycle.
- We will stick to COMET method while discussing the notations and terms of UML. The UML standard we are covering in this book is UML 2.

☞ **Diagrams which are supported by UML notation**

- Following are the diagrams which are supported by UML notation:
  1. Use Case diagram
  2. Class diagram
  3 Object Diagram
  4. Communication Diagram
  5. Sequence Diagram
  6. State Machine Diagram
  7. Activity Diagram
  8. Composite structure diagram
  9. Deployment Diagram

## Syllabus Topic : COMET Based Software Lifecycle

### 1.7.2 COMET Based Software Lifecycle

➔ **(Feb. 2016)**

Q. Explain COMET and phases of COMET.
**SPPU- Feb. 2016(In sem), 5 Marks**

UML based software modeling and architecture design method is often called COMET (Concurrent Object Modeling and Architecture Design Method). This is specification for single system and covers the entire project development life cycle.

☞ **Different phases of COMET**

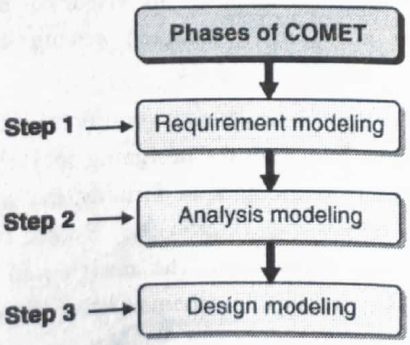


**Fig. 1.7.1 : Phases of COMET**

**1. Requirement Modeling Phase**

In requirement modeling functional requirement of the system is defined in terms of actors and use cases. Also stories or narrative description of each use cases are written.

⬇

**2. Analysis Modeling Phase**

Static and dynamic model of the system is developed during analysis modeling phase. Static modeling involves class diagram and the relationships. Dynamic modeling captures the dynamic changes of objects like collaboration diagrams.

⬇

**3. Design Modeling Phase**

Entire software architecture is developed during modeling phase. It also involves applying coding strategies and architectural design patterns. Encapsulation, inheritance, polymorphism etc. are implemented in this phase.

## Syllabus Topic : Reusability

### 1.7.3 Reusability

Q. What is the importance of use case in designing ?
**(2 Marks)**

- UML (Unified modeling Language) enforces the reusability in software development process.
- Reusability can be implemented at various levels in design phase various design tools.
- Object oriented mechanism support the reusability mainly though inheritance of components. However UML support various tools that can be reused at many places in design where common functionalities are required.
- Use case components can also be reused at different area in design.
- Reusability will be explored in next section for use cases where we will study the use case relationships.
- Two main components of use cases where reusability can be visualized are: include and extend relationship. These two components will explored in upcoming section.

## 1.8 Use Cases

UML supports both the aspects of modeling: static and dynamic. In previous section overviewed the static object modeling which captures static information of the system at beginning. Dynamic modeling captures runtime behaviour of the system. There are notations and diagrams that very efficiently describes the run time changes of the system's state. We will start exploring the notations and diagrams one by one.

### 1.8.1 Introduction to the Use Case

**Q. Explain the basic terminologies and notation related to use cases. (4 Marks)**

- Use cases capture the functionality of the system from user's perspective. It is represented as the sequence of interaction between **actor** and the **system.**
- Use case diagram was proposed by Jacobson in the book subtitled **"A use case driven approach"** 1992.
- Use cases are described in texts as well in diagrams. Text form of use cases are called **use case descriptions.**
- Textual **use case description** describes the interaction between user of the system also called *actors* and high level function within the system also called *use case*. Each interaction scenario can be specified in step by step manner.
- The system is depicted as it is visible to the user. Internal working and processing of the system is not highlighted in use cases.
- Apart of textual description, most visible part of use cases is the use case diagram which shows three aspects of the system : 1. Actors 2. Use Cases 3. System Boundary or Sub-system boundary. Fig. 1.8.1 shows the general notations of Use case diagram.

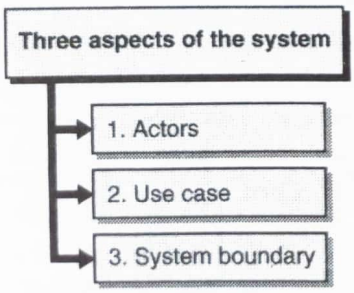


**Fig. C1.8 : Three aspects of system**

➜ **1. Actors**

- Actors are represented by arranging sticks in human shape. But not necessarily it represents the human user.
- It can be used to represent user, other system or devices that interacts with particular high level functionality (use case) of the system.

➜ **2. Use case**

- Use case basically represents a high level functionality of the system.
- Filling an online application in a web based system is one use case. Use case is represented by ellipse inside a box.

➜ **3. System boundary**

- Use cases are packed inside a box called the system boundary. Actors are outside the boundary.
- System boundary is used to show the enclosure of system functionality.

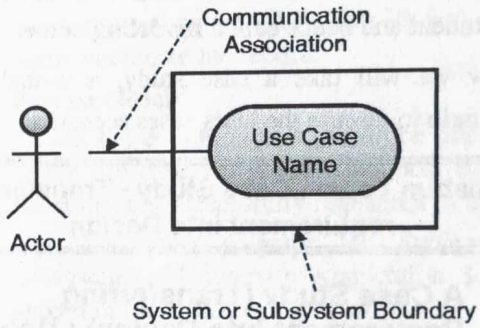


**Fig. 1.8.1 : Notation of use case diagram**

### 1.8.2 Use Case Actors

**Q. Explain use case actor. Who are primary and secondary actors? Explain with example. (4 Marks)**

Use case actors are the external user of the high level functionality (use case) of the system. Actors are outside system's boundary and hence external to the system. It often depicts the role played by external entities which interacts with the system. We should however not confuse with actual user of the systems and the actors. Actors can be used as a role player for different same kind of users. So the actors could be an actual system user, another system or the device. But most often you will find a human user as an actor. In real time applications, entities like timers, sensors, actuators interact with the system functionality. So they play the role of actor. There are few important points to remember while talking about actors.

- Actors are external entities that interact with the system. Commonly, actors are human users of the system. But it can also be external interfaces to the system, other system, devices like actuators, sensors, timers etc.
- The use case is initiated by **primary actor**. The primary actor activates the system by providing it the first input and after the system starts responding.
- After that all the other actors can interact with the system. Other actors are called **secondary actors**.
- It is possible that one actor is primary actor in particular use case and in another use case, same actor is secondary actor.
- Often an actor that is actually a device but represents the action of a human actor. For example, attendance marking device is used by students to mark their attendance in a University Application Software. Then this scenario, attendance marking device plays the role of student and hence called **modeling actor.**
- Now we will take a case study, a virtual practical scenario to discuss the Uses cases scenarios.

## Syllabus Topic : Case Study - Transferring requirement into Design

### 1.8.3 A Case Study (Transferring Requirement into Design) : University Application Software

- Now we will try to discuss each and every aspect of use cases in relation with a practical scenario. We are taking a case study of university application.
- We will be considering a virtual University for which we want to prepare a web based application with several functionalities. It will maintain student and teacher activities and accessible remotely.
- It will complete solution for automation of all academic activities involving student ID-card, attendance, examination, admission, leaves, notices, feedback etc.
- The obvious choice for design methodology will be object orientation as it has clear real world problem domain. It has been experienced and realised over the years that real world automation problems are best addressed by object oriented methodology.
- Understanding the actual process of University activities, identifying the objects in problem domain are few primary steps that we have to perform in object oriented analysis.
- We will be doing object modeling in next section, in this section we will concentrate on identifying different use case scenarios.
- For simplicity, we will be considering following required services needed by the University Application. We have assigned these activities to dedicated **subsystems** :
  1. Student_Id_Card
  2. Faculty Id-Card
  3. Lecture Updates
  4. Attendance
  5. Examination
  6. Leave
  7. Notices
  8. Feedback
  9. Application Administration
- As we will discuss different notation and terms of object oriented designing and analysis, we will keep on refining our virtual University Application requirements.
- So for now we are not freezing all the requirements of University Application. As we will progress, will specify the requirements, use case scenario and will explore it.

## Syllabus Topic : Use Case Modeling

### 1.8.4 Use Case Modeling : A Simple use Case for University Application

**Q.** What are the common essentials of use cases need for formal description of use cases? **(6 Marks)**

- Now we will consider a simple use case for creating student record in Student_Id_Card subsystem of the University Application Software.
- The possible high level functionalities offered to user from the subsystem named Student_Id_Card is listed below. These are basically subsystem features visible to the user and can be listed as identified **Use Cases** for **Student_Id_Card**.
  (a) Create Student Record
  (b) Approve Student Record
  (c) Generate Student ID-card

Use case should have description of some basic information called the essentials of the use case. The common essentials are listed in Fig. C1.9.

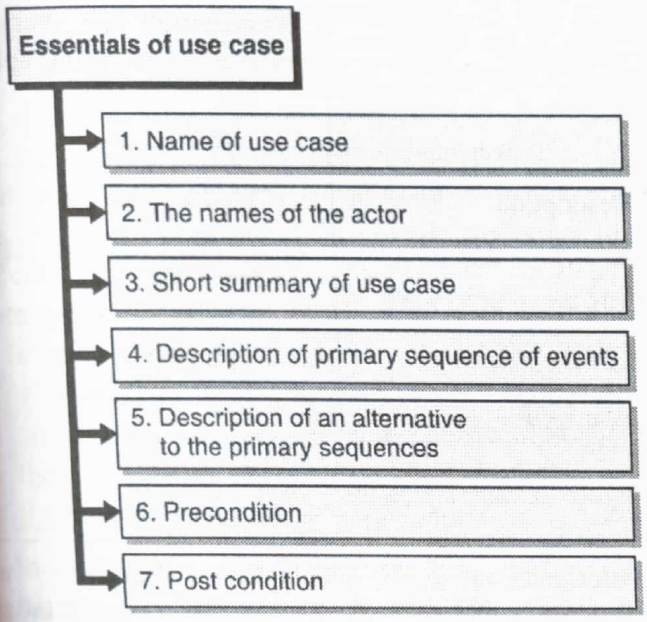


**Fig. C1.9 : Essentials of use case**

➜ **1. Name of use case**

- In University application, Create Student Record is one Use case for Student_Id_Card subsystem. There could be more than one use cases in given subsystem.
- Other identified use cases are: Approve Student Record and Generate Student Id-Card.

➜ **2. The names of the actor**

- For Student_Id_Card subsystem , the actors are identified for each use case.
- For example, Creating Student Record is restricted to students and only they can create the initial record by clicking on New User/Register link. So the only actor for Create Student Record is student (Except for one alternate case that we will mention later). Since the activity performed by Student causes the subsequent activities like approval and Id-card generation, student can termed as **primary actor**.

➜ **3. Short summary of use case**

- A short description of the functionality from user's perspective should be there.
- **Example :** The use case Create Student Record can have short description like : A student can create the individual record by clicking on New User/Register link on home page and filling up the registration form by entering a guest username password provided by university.

➜ **4. Description of primary sequence of events**

- This is the general sequence of events within the use case.
- **Example :** Firstly user opens the home page and clicks on New User / Register link. Then provides a temporary username and password and after that can fill the form, upload the photograph and submit.

➜ **5. Description of an alternative to the primary sequences**

- This is general description of sequence for the alternate to primary sequence.
- **Example :** In our University Application, if students do not have guest login and password, they directly approach application admin to create the account.

➜ **6. Precondition**

- Conditions that must be true before starting the use case.
- **Example :** The Precondition for Create Student Record is student should have a temporary guest username and passwords which student can use to login and create the record.

➜ **7. Post condition**

- This is the condition that should be true after end of the use case. So the obvious post condition is the student is successfully registered in the system.
- The Use case diagram for Student_Id_Card subsystem of University Application Software is shown in Fig. 1.8.2.
- The actors interacting with system will be Student and Application Administrator. Box shows the sub system boundary and ellipses are the uses cases.

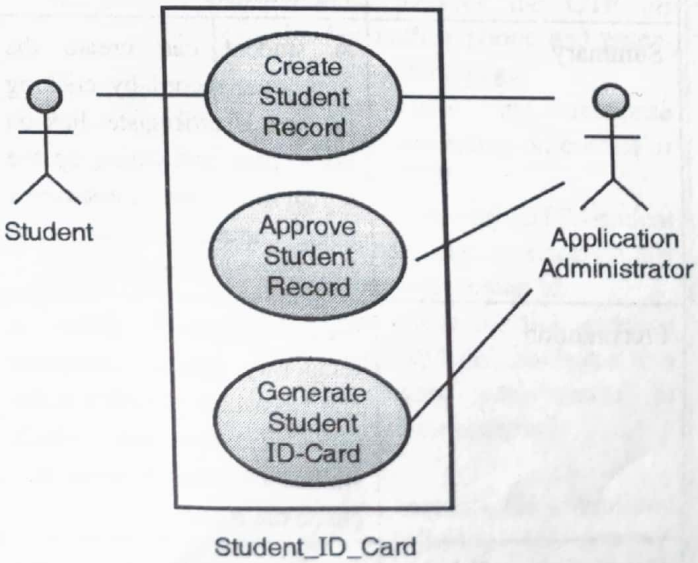


**Fig. 1.8.2 : Use case for student_id_card supsystem of university application software**

**Syllabus Topic : Use Case Template**

## 1.8.5 Use Case Documentation Model and Template

**Q.** Consider the following scenario :
As your academic project, you have been assigned the task for designing and implementing the University application software. One module or subsystem called Student_ID_Card subsystem is responsible maintaining student's record. A student can create record and submit it in the system. An applications administrator examines and verifies the record and generated the ID card.

a. Design the possible uses cases. (You can make sensible assumption while designing)

b. Give formal description of the uses with including following terms :

1. Pre and post condition
2. Primary and Alternative sequence steps
3. List of actors
4. Summary of use cases **(8 Marks)**

– The description of use case is represented in a formal structured documentation. We will take one use case from our University Application and try to represent in well structured manner and will try to cover all 8 points (essentials) introduced in last section.

– We will be taking the use case "Create Student Record" and document it is shown in Table 1.8.1.

**Table 1.8.1**

| Use Case Name | Create Student Record |
|---|---|
| Actors | Student, Application Administrator |
| Summary | A student can create the individual record by clicking on New User/Register link on home page and filling up the registration form by entering a guest username password provided by university. |
| Precondition | Student should have a temporary guest username and passwords provided at the time of admission, which student can use to login and create the record. |
| Description Primary Sequence | 1. User opens the home page and clicks on New User / Register link.<br>2. Then provides a temporary username and password and after that can fill the form, upload the photograph and submit. |
| Description of alternative Sequence | 1. Student does not have guest login and password; He/She directly approach application admin to create the account.<br>2. Application admin fill the student's record on students' behalf. |
| Post Condition | Students details are successfully recorded in the system. |
| Non Functional Requirement | There should be proper validation of student's record before storing in the system. Data should be securely stored. |

– Now similar tables can be formed to model the use case description of other two use cases namely "Approve Student Record" and "Generate Student Id-Card ". It will be good practice to think about the sequence, actors and description for the other two use cases. We are leaving it for exercise.

## 1.8.6 Use Case Relationship

➔ **(Feb. 2016)**

**Q.** What is extends and include stereotype in use case diagram ? Explain with an appropriate example.
**SPPU - Feb. 2016(In sem), 5 Marks**

In last section, we explored one use case scenario for University application i.e. Create Student Record. Performing further investigation on this use case may raise a conclusion that it is too abstract to show it as single use case. Creating Student Record itself includes more use cases or may depends on other use case that should be included in uses case diagram for complete understanding of the scenario. The dependencies and relationship of the use cases can be described by two standard terms called **include** and **extend.**

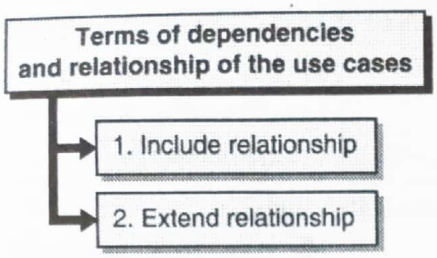


**Fig. C1.10 : Terms of dependencies and relationship of use cases**

### 1. Include relationship

➔ **(May 2016, April 2017)**

Q. Diagrammatically show include relationship on the context of use case diagram.
**SPPU - May 2016, 2 Marks**

Q. What is include stereotype in use case diagram.
**SPPU - April 2017(In sem), 1 Mark**

**Inclusion use case** is the use case having some functionality that can be used by other use cases. Several use cases can include that common functionality by extending to an inclusion use case.

For example, in our University application, student's mobile number is validated using OTP when student performs **Create Student Record**. Another use case named **Generate Id Card** is performed by Administrator but in case of lost identity card, student can interact with **Generate Id Card** by verifying mobile number using OTP and then student can generate a duplicate ID card.

So we can identify a new use case **Validate Using OTP** that can be included in both, **Create Student Record** and **Generate Id Card**. Fig. 1.8.3 shows this inclusion in use case diagram.

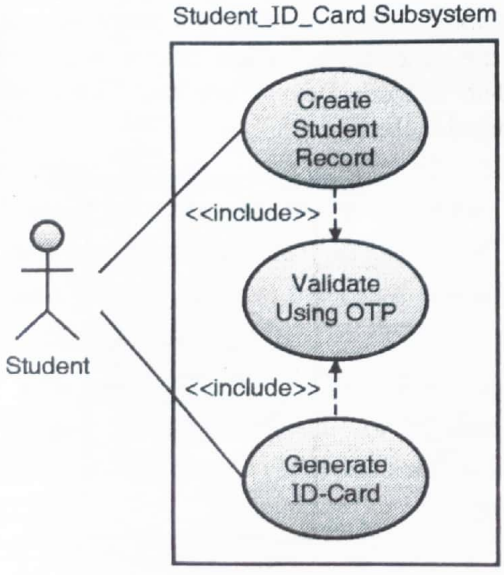


**Fig. 1.8.3 : Example of Inclusion relationship**

- Here Validate Using OTP is termed as inclusion use case while the other two are base use cases or executable use cases.
- A formal description of the Use case Validate using OTP is provided in Table 1.8.2.

**Table 1.8.2**

| Use Case Name | Validate using OTP |
|---|---|
| Summary | Validation of student mobile number and email Id before submitting the record for approval |
| Actor | Student |
| Precondition | Case1 : Students record is not submitted for approval and mobile number and email id need to be verified<br>Case 2 : Student record is already approved but it is not verified that an authentic student is requesting duplicate Id- card |
| Primary Sequence | **Case 1**<br>1. Student fills up the form for personal record including mobile number and email id. Student submits the form.<br>2. After submitting, student navigates to a new web page. New web page asks for OTP received on mobile number.<br>3. System generates the OTP and sends it to student's mobile phone.<br>4. Student receives the OTP on his/her mobile phone and enters it in requesting page.<br>5. Student gets the response message depending on correct or incorrect OTP.<br>6. In case of correct OTP, student gets a message to click on the link sent to the email id.<br>7. Student clicks the link received on email and link navigates to a page saying your record is submitted for approval.<br>**Case 2**<br>1. Student requests for a duplicate id by clicking on generate duplicate Id-card.<br>2. After clicking, student navigates to a new web page. New web |

| Use Case Name | Validate using OTP |
|---|---|
| | page asks for OTP received on mobile number.<br>3. System generates the OTP and sends it to student's mobile phone.<br>4. Student receives the OTP on his/her mobile phone and enters it in requesting page.<br>5. Student gets the response message depending on correct or incorrect OTP.<br>6. In case of correct OTP, student gets a message to click on the link sent to the email id.<br>7. Student clicks the link received on email and link navigates student to a new page where student can print the duplicate ID card. |
| Alternate Sequence | 1. Student is navigated to the page where he/she has to submit the OTP.<br>2. Student has not received the OTP and after timeout period student clicks on link Problem in Receiving OTP.<br>3. After clinking the link student verifies the email id as specified in primary sequence.<br>4. After verifying email id , student gets the message that, form has been submitted for approval and but Mobile Number is not verified.<br>5. System marks a flag in the student's record that Student's phone number is not verified.<br>6. Admin follows the manual process for verifying student's mobile number. |
| Post Condition | Student's record gets submitted successfully for admins approval with auto verification of valid mobile number and email id. |
| | In an alternate case, record is submitted but marked as unverified mobile number. |

## 2. Extend Relationship

**(May 2016, April 2017, May 2017)**

**Q.** Diagrammatically show extend relationship on the context of use case diagram. **SPPU - May 2016, 2 Marks**

**Q.** What is extends stereotype in use case diagram ? **SPPU - April 2017 (In Sem), 1 Mark**

**Q.** Elaborate extend include with the context of Use case diagram with example. **SPPU-May 2017, 4 Marks**

- It has been illustrated in previous section that some use case scenario cannot be represented in abstract and has to be expanded.
- One form is inclusion where common functionality is included in one or more than one use cases.
- Another is extension where a use case can have various alternative extended functionalities and each functionality can be represented as separate use case.
- The abstract use case that needs to be extended is called base use case. The use cases that represents extended functionality is called extended use case.
- To understand the extend relationship, we well further explore the requirements for University Application. Now, consider the following requirement for Lecture_Update sub system.
- University has recently decided to keep the live updates of attendance and lectures of every college. Faculties are required to fill the lecture details and attendance after each lecture on University application by interacting with "Lecture Updates" sub system use case.
- The above requirement seems pretty straight forward for use case design where faculty will interact with Update Lecture Details use case. This use case is depicted in the Fig. 1.8.4.

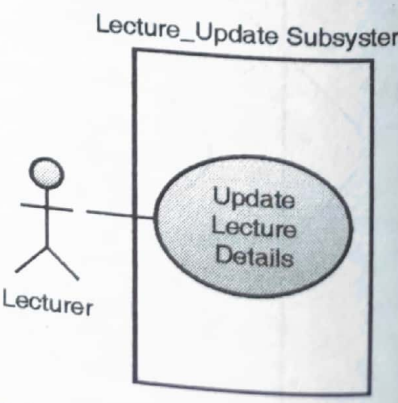


**Fig. 1.8.4 : Use case for update lecture details**

- University has realized that every time it is not feasible for lecturer to mark entire attendance on web interface on University portal after each lecture.
- So they have provided low cost thumb devices that can easily mark the attendance and transfer it over the air (OTA).
- Student can quickly mark the attendance by thumb impression and entire attendance will get transferred over the air (OTA) to server which hosts Attendance subsystem.
- Now our abstract use case "Update Lecture Details" has two extended alternatives.
  (A) One is conventional way of marking attendance where actor is Faculty and he/she interacts with web interface of Attendance subsystem to fill attendance. We call this use case as "Mark Online Attendance"
  (B) One is the use case where actor is student (or mobile thumb device playing the role of student) and attendance is transferred wirelessly to Attendance subsystem through well defined port and interface. We call it OTA attendance and named the use case as "Mark OTA Attendance".
- So far, there are three identified use cases for this scenario :
  1. **Base use case :** Update Lecture Details
  2. **Extended use case :** Mark Online Attendance
  3. **Extended use case :** Mark OTA Attendance
- Now another related terminology is **"Extension Point"**. This is used to represent the precise location in base use case where extensions are added.
- A name should be given to each extension point, and it should be inserted into base case from where extensions are forked or added.
- In Update_Lecture subsystem of University Application, base use case is Update Lecture Details. So "Mark Attendance" can be taken as sensible extension point from where two extensions are forked: Mark Online Attendance and Mark OTA Attendance.
- The three uses cases, their extension relationship, actors and extension points for Update_Lecture subsystem of University Application Software is shown in Fig. 1.8.5.
- Notice that Update Lecture Details use case ellipse is divided by a line the accommodate extension point.

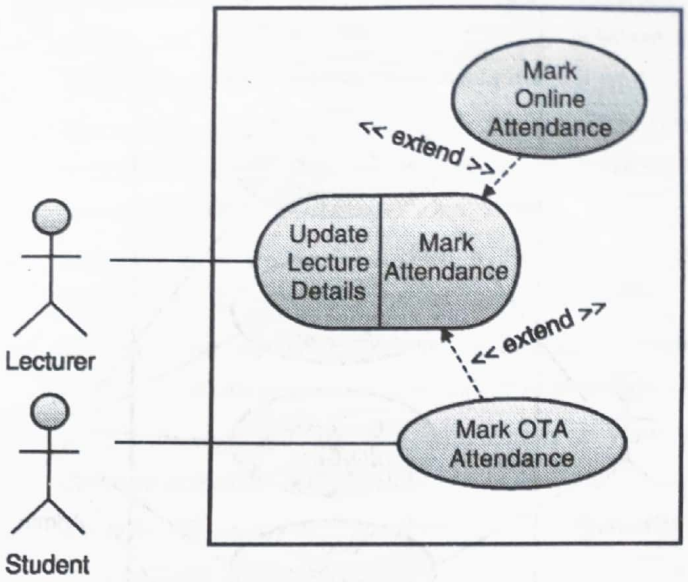


**Fig. 1.8.5 : Use case diagram for extend relationship in update lecture details**

## 1.8.7 Use Case Packages

**Q.** Why it is good practice to group use cases in a use case package? Explain with UML notation. **(4 Marks)**

- A large application software project like University Application can have large number of possible use case scenarios. Different subsystems like Attendance, Admission, Leaves, Examination has different set of use case scenarios.
- **Use case packages** are used for grouping logically similar kind of use cases at one place to handle the complexity.
- A use case package therefore represents the high level functionality of the system and contains a number of more specific use cases related to the high level functionality.
- Sometimes use case packages are formed base on the actor that interacts with system. So it is possible to have a dedicated package for Student's uses cases and so for others.
- Fig. 1.8.6 shows the Use Case package for Student_Id_Card subsystem. The name of package is "StudentRecordManagement" and the primary actor for this package is Student. Other actor is Application Administrator.

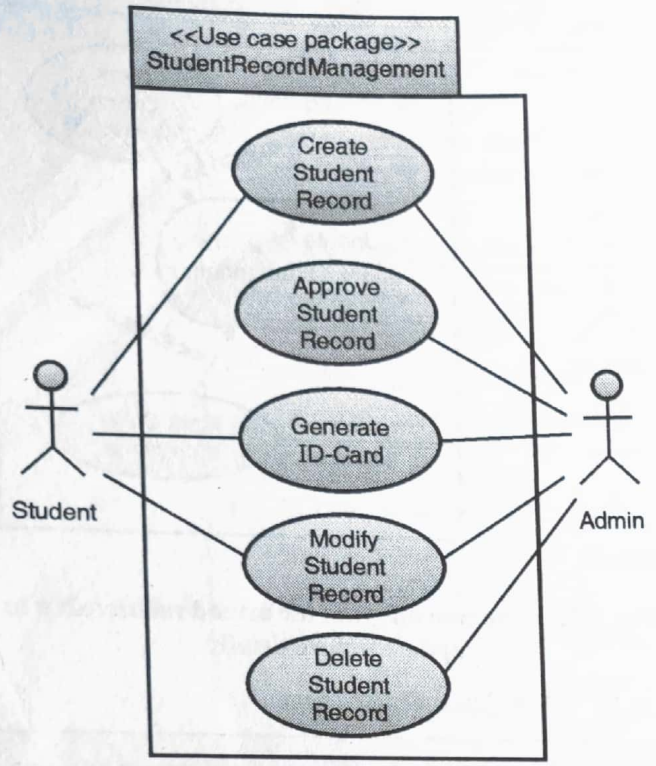


**Fig. 1.8.6 : Example of use case package**

**Ex. 1.8.1**

Draw the use case diagram for an e-learning system where users of the system will be system administrator, course creator and students. Administrator will manage the system and start the initial system setup and configuration. Creator will create the video lectures for the courses assigned by administrator. Students can register, login download and subscribe the video lectures by paying online.

**Soln. :**

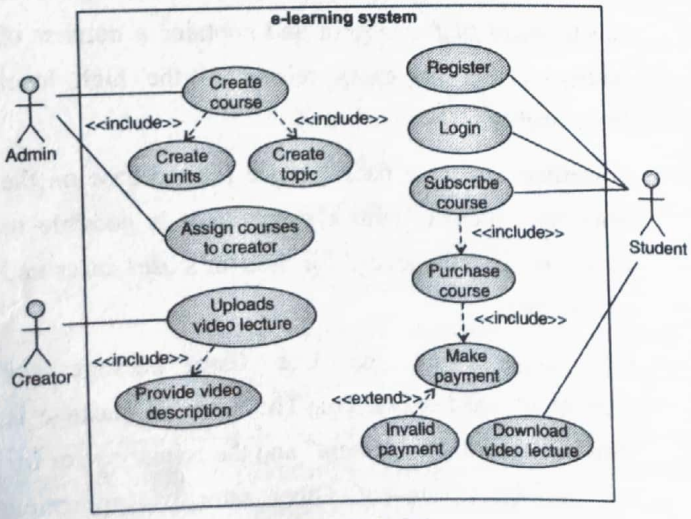


**Fig. P. 1.8.1**

## Summary

- Design process in software projects are very well defined and need significant effort before starting the actual implementation or coding. The approach of design is often termed as "**Model based Software Design and Development**".
- **Software design** is often considered as initial steps towards the solution with available capabilities. There could be multiple designs for the same software system covering different aspects of the system to be developed.
- The term **Software design** is used to refer both, the **process** of designing and the design product or the **model**.
- **Design process** is where the creativity lies. An experienced and skilled architect needs to be creative also in the modern days of competitive software development strategies.
- A **model** is nothing but the architecture that describes all aspects of the system that we want to build.
- The outcome of software design is often classified in three levels :
  1. Architectural Design
  2. High Level Design
  3. Detailed Design
- The highest level of abstraction of the software system can be represented by **architectural design**. We can have a very high level view of the entire system and can observe it as many components interacting with each other.
- Breaking the architectural design at component level yields the **high level design.**
- **Detailed designing** is the boundary where we actually switch from deigning to implementation. Each and every module structure is implemented here. The interface to the module is identified.
- In early 1970's structured programming and designing starts getting popularity among software developers. This kind of designing methodology was based data-flow oriented designing.
- The objective of **structured designing** was to organize the entire system in well defined modules. The structured designing was one of the first well documented designing methodologies.

- Two important approaches of designing were introduced with structured designing. They were called coupling and cohesion, two important aspects of modularity.
- **Procedural design** is even more conventional design methodology where design decisions are made even when the system functionality is not understood properly or even when all the information is not available.
- One of the most popular design methodology widely practised from desktop, server to even embedded systems is **Object oriented design and analysis**.
- **Object orientation** is way of thinking a problem with real world perspective. The concept of the system in mind is mapped in the objects, their properties behaviour and interaction.
- **Object orientation** avoids the problems and pitfalls associated with conventional designing methodology.
- The concepts that we mostly address in **object orientation** approach is classes, objects, inheritance, generalization, specialization, polymorphism and abstraction.
- **Objects** are the central concept in object orientation. It represents the abstraction of any real object in problem domain. It reflects the capability of the system. It reflects what information system will keep and also it reflects what methods it offers to interact with the system.
- Information hiding or **abstraction** is another central approach of object oriented designing. If a module is well contained with its own information, it will be less dependent on other modules and hence changes can be easily done without affecting the other modules.
- **Object orientation analysis** is the first method applied in object oriented methodology of software development. This is the analysis phase of software development. The mapping of real world to programming world is performed here. Objects are identified along with their properties and behaviour. The phase is often called **object modeling**.
- **UML** (Unified modeling Language) is standard modeling language and notation for object oriented analysis and design.
- **OMG** (Object management group) has adopted UML as standard notation for designing in 1997. The most significant contributors in forming the notations and specifications of UML are Grady Booch, Ivar Jacobson and James Rumbaugh. The design and notation of UML was developed by them at Rational Software.
- **Use cases** capture the functionality of the system from user's perspective. It is represented as the sequence of interaction between **actor** and the **system.**
- Use case diagram was proposed by Jacobson in the book subtitled *"A use case driven approach"* 1992.
- Use cases are described in texts as well in diagrams. Text form of use cases are called ***use case descriptions.***
- **Activity diagrams** are used to represent flow of different activities in the system or sub system.
- **Activity diagrams** are standard UML diagrams and have well defined notations.
- **Activity diagrams** shows the flow of control and sequencing among different possible activities.
- **Activity diagrams** do not explore system design or system's processing logic, instead focuses on flow of activities that a user of the system can experience.

## 1.9 Exam Pack (University and Review Questions)

☞ **Syllabus Topic : Introduction to Software Design**

**Q.** What is the role of designing in any system or project implementation? Explain model based software design and development. *(Refer section 1.1.1)* (6 Marks)

**Q.** What is software design? *(Refer section 1.1.2)* (4 Marks)

**Q.** Describe classification of software design based on abstraction level ? *(Refer section 1.1.2)* (4 Marks)

**Q.** Explain the fundamental design concepts applied in modern software design. *(Refer section 1.1.2)* (6 Marks)

☞ **Syllabus Topic : Procedural and Structural Design Methods**

**Q.** What are structured and procedural design methods ? *(Refer section 1.2.2)* (6 Marks)

☞ **Syllabus Topic : Object Oriented Analysis and Design Methods**

**Q.** Explain the concept of object oriented analysis and designing and its role in modern application software development. *(Refer section 1.2.3)* (6 Marks)

☞ **Syllabus Topic : Requirement vs Analysis vs Architecture vs Design vs Development**

**Q.** Compare requirement and analysis. *(Refer section 1.3.1)* (4 Marks)

**Q.** Differentiate analysis and architecture.
*(Refer section 1.3.2)*(4 Marks)

**Q.** Compare architecture vs design.
*(Refer section 1.3.3)*(4 Marks)

**Q.** Compare design and development.
*(Refer section 1.3.4)* (6 Marks)

### Syllabus Topic : 4+1 Architecture

**Q.** What is 4+1 architecture view model? Explain with suitable diagram. *(Refer section 1.4)* (5 Marks)
**(Dec. 2016)**

**Q.** Explain different views of software architecture.
*(Refer section 1.4)* (4 Marks)

### Syllabus Topic : Introduction to UML - Basic Building Blocks

**Q.** What is Unified Modeling Language (UML)? List few standard notation and diagrams of UML.
*(Refer section 1.7.1)* (4 Marks)

### Syllabus Topic : COMET Based Software Lifecycle

**Q.** Explain COMET and phases of COMET.
*(Refer section 1.7.2)* (5 Marks) **(Feb. 2016(In sem))**

### Syllabus Topic : Reusability

**Q.** What is the importance of use case in designing ?
*(Refer section 1.7.3)* (2 Marks)

**Q.** Explain the basic terminologies and notation related to use cases. *(Refer section 1.8.1)* (4 Marks)

**Q.** Explain use case actor. Who are primary and secondary actors? Explain with example.
*(Refer section 1.8.2)* (4 Marks)

### Syllabus Topic : Use Case Modeling

**Q.** What are the common essentials of use cases need for formal description of use cases?
*(Refer section 1.8.4)* (6 Marks)

### Syllabus Topic : Use Case Template

**Q.** Consider the following scenario :
As your academic project, you have been assigned the task for designing and implementing the University application software. One module or subsystem called Student_ID_Card subsystem is responsible maintaining student's record. A student can create record and submit it in the system. An applications administrator examines and verifies the record and generated the ID card.

a. Design the possible uses cases. (You can make sensible assumption while designing)

b. Give formal description of the uses with including following terms :
   1. Pre and post condition
   2. Primary and Alternative sequence steps
   3. List of actors
   4. Summary of use cases

*(Refer section 1.8.5)* (8 Marks)

**Q.** What is extends and include stereotype in use case diagram. Explain with an appropriate example.
*(Refer section 1.8.6)* (5 Marks) **(Feb. 2016(In sem))**

**Q.** Diagrammatically show include relationship on the context of use case diagram.
*(Refer section 1.8.6)* (2 Marks) **(May 2016)**

**Q.** What is include stereotype in use case diagram.
*(Refer section 1.8.6)* (1 Mark) **(April 2017 (In sem))**

**Q.** Diagrammatically show extend relationship on the context of use case diagram.
*(Refer section 1.8.6(2))* (2 Marks) **(May 2016)**

**Q.** What is extends stereotype in use case diagram?
*(Refer section 1.8.6(2))* (1 Mark) **(April 2017(In sem))**

**Q.** Elaborate extend include with the context of Use case diagram with example.
*(Refer section 1.8.6(2))* (4 Marks) **(May 2017)**

**Q.** Why it is good practice to group use cases in a use case package? Explain with UML notation.
*(Refer section 1.8.7)* (4 Marks)

❑❑❑

UNIT II

CHAPTER 2

# Static Modelling

**Syllabus Topics**

Analysis Vs Design, Class diagram - Analysis - Object and classes finding analysis and Design - design classes, refining analysis relationships, Inheritance and polymorphism, Object diagram, Component diagram - Interfaces and components, deployment diagram, Package diagram.

**Syllabus Topic : Analysis vs Design**

## 2.1 Analysis vs Design

– Analysis phase concentrates on problem domain and clarifies all the related concepts of problem. This helps in starting the solid design activity. Design phase starts with exploring the solution domain.

☞ **Difference between the analysis and design**

**Q.** Compare analysis and design. **(4 Marks)**

| Sr. No. | Parameters | Analysis | Design |
|---|---|---|---|
| 1. | Domain | The analysis phase explores the problem domain and applies the Unified Process and tools to clarify the requirements. | The design phase explores the solution domain and applies the Unified Process and tools to formulate the system design. |
| 2. | Involvement | Analysis is conducted by system analysts, business analysts or system etc. | Design formulation is conducted by system architects, developers or system analysts. |
| 3. | Outcome | The outcome of analysis is software requirement specification document. | The outcome of designing phase is low level system design or detailed design document. |
| 4. | Components | The primary components formulated during the analysis phase is requirements specification for different system modules, their descriptions, use cases, use case stories, analysis classes etc. | The primary components formulated during the design phase and static and dynamic view of system in the form of class and object diagram, collaboration diagram, state diagram, component and deployment diagram etc. |
| 5. | Stakeholders | System analysts, customers, end users, business analysts, product owners are the primary stakeholders in design. | System architects, system analysts, developers, project managers are the primary stakeholders in design phase. |

**Syllabus Topic : Class Diagram**

## 2.2 Class Diagram

### Class

**Q. What is class? (2 Marks)**

In static modeling, we capture static view of the system. Structure of the system that does not change with time is modelled in static modeling. The concepts of problem in hand are translated into designing constructs called **classes.**

### Objects

**Q. What is object? (2 Marks)**

A system architect starts with analysing the requirements and identifying the use cases. Then **objects** are identified in problem domain keeping the use cases in mind. The objects are identified with its properties and capabilities. The actions of capabilities are not taken into consideration however in static modeling. Classes are used to declare the properties and behaviour of the object.

### Importance of class diagram

→ **(May 2016)**

**Q. Why is a class diagram important in static modeling? SPPU - May 2016, 5 Marks**

- Class represents the abstraction of real world object.
- Objects of similar kind belong to one class. Collection of objects of similar characteristics is class. It captures static structure of the system.
- In programming languages, class is prototype or blueprint using which objects are created. We can treat classes as user defined data type.
- In designing, the concept are mapped in objects and objects are statically represented as classes. For example Student is class. Subject is a class. Objects are nothing but instances of the class.
- Objects on the other hand, is not a concept or declaration but an existing "thing" that posses some characteristic and behaviour. Object belongs to some class that is already declared and understood.
- The properties or attributes held by object is represented in class as variables.
- Behaviour of object is represented in class as operations or methods.
- Operations declared in classes are specification of the function that objects can perform.

### Two important constituent of classes

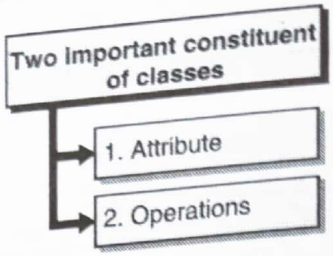


**Fig. C2.1 : Two important constituent of classes**

→ **1. Attribute**

Representing the properties of object.

→ **2. Operations**

Representing the functions that object of the class type can perform.

### Definition of class diagram

**Q. What is class diagram? (1 Mark)**

- In UML, classes are simply represented as boxes. One or more arrangement of classes with their possible relationship is called class diagram.

### Simple representation of classes and objects in UML

**Q. Explain attributes and operation and their representation in class diagram. (4 Marks)**

- Fig. 2.2.1(a) shows the simple representation of classes and objects in UML.

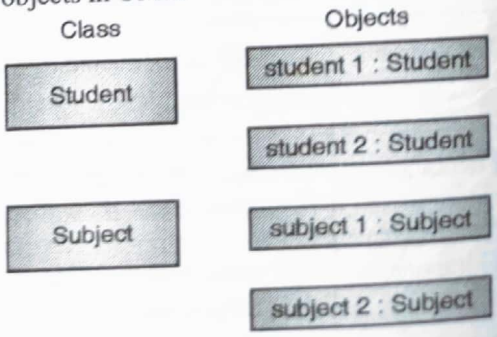


**Fig. 2.2.1(a) : Simple representation of classes and objects in UML**

- Fig. 2.2.1(b) shows the simple representation of classes and objects in UML with properties or attributes.

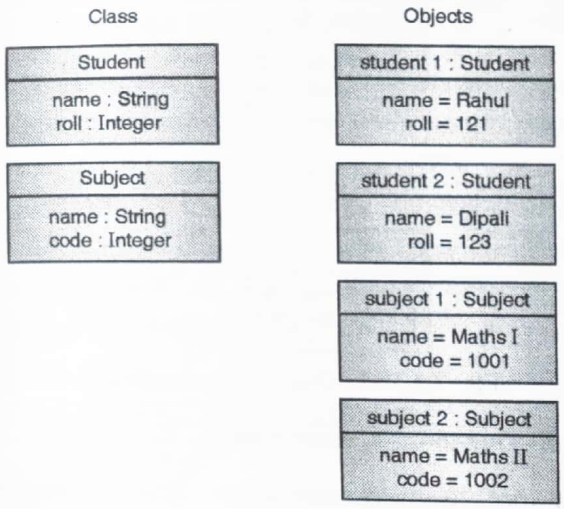


**Fig. 2.2.1(b)**

- Fig. 2.2.1(c) shows the simple representation of classes and objects in UML with properties or attributes and operations. Operations reflect the capabilities of the object and task that it can perform.

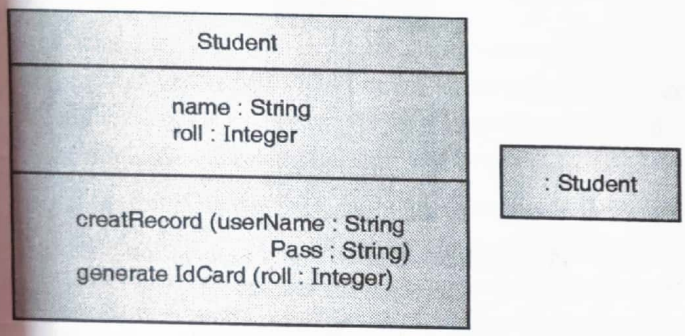


**Fig. 2.2.1(c)**

☞ **Use of class diagram**

- Class diagrams are used to form the static model of the system. A class diagram can be more precisely described by showing the relationship between the classes.

☞ **Three type of relationships**

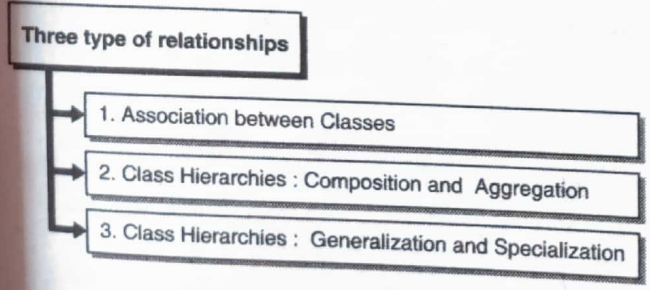


**Fig. C2.2 : Three types of relationships**

## ➜ 2.2.1 Association between Classes

**Q. What is association relationship between classes? Take a suitable example and draw the class diagram. (4 Marks)**

- The static relationship between the classes is depicted by associations. For example if Student *registers for* particular Subject , then the association relationship is represented by term "registers for" and "*Register*" is an association.
- Student and Subject are classes.

☞ **Link**

- **Link** is used to represent the association between the classes or objects. Link represents an instance of association.
- For example, if Rahul registers for Maths1 then the link exist between Rahul and Maths1 showing an association between the classes Student and Subject.

☞ **Binary associations**

- There is bidirectional relationship between the classes which are associated.
- For example, Student **registers** *for* Subject is one way of thinking the association and another is Subject **enrols** students.
- In most of the cases, association are between two classes and so called **binary associations**. There could be unary associations also that are the association with class itself or self association.
- There are relations possible between more than two classes. For example ternary associations or higher order associations.

☞ **Associations in class diagrams**

- Associations can be depicted as lines or arcs between two classes in class diagram. Name of the association is mentioned with the line or arch connecting two classes.
- One binary association is shown in Fig. 2.2.2 which shows the association Subject is taught by Subject Teacher.
- While reading the relationship, it is read from top to bottom or from left to right. For example, in the Fig. 2.2.2, we can read the relation as "Subject is taught by Subject Teacher" and one instance of the relation can be stated as "Maths1 *is* **taught by** Prof. Anil.

- In large class diagrams, where relation exists between many classes, it is not possible to arrange the classes in top bottom or left right fashion. So in that case arrow mark denotes the direction of association to be read.

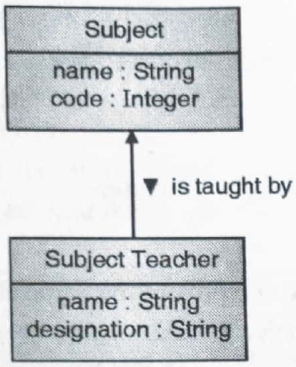


**Fig. 2.2.2 : Example of association relationship between two classes**

## 2.2.1(A) Multiplicity of Association

- One important aspect of association is multiplicity that defines how many instances of one class is associated with single instance of another class.

☞ **Types of multiplicity of association**

**Q.** Explain different degrees of multiplicity of association between classes with suitable example.
a. One-to-One Association
b. One-to-Many Association
c. Many-to-many Association **(6 Marks)**

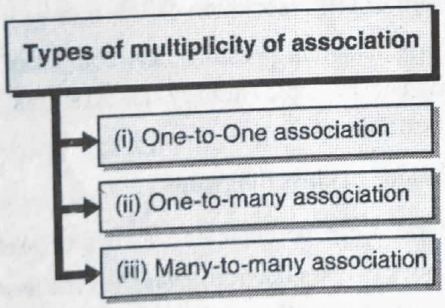


**Fig. C2.3 : Types of Multiplicity of Association**

➜ **(i) One-to-One association**

- The two classes are related and have one to one associative relationship. It means there is one to one relationship in both directions. Exactly one instance of a class is related to one instance of another class.
- **Example :** Example is the relationship between Principal and College. So a college has only one principal and one principle can administer only one college.

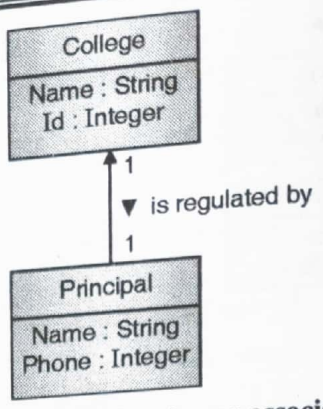


**Fig. 2.2.3(a) : One to one association**

➜ **(ii) One-to-many association**

- In one to many association, there is one-to-one association in one direction and in opposite direction there is one-to-many association between the classes.
- **Example :** Student studies in only one college but one college has many students. Fig. 2.2.3(b) shows the concept.

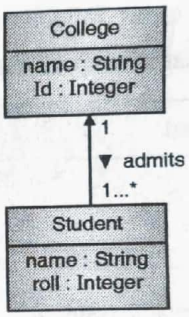


**Fig. 2.2.3(b) : One to many association between the classes**

➜ **(iii) Many-to-many association**

- Many to many association is used to describe the association between two classes where there is one-to-many relationship in each direction.
- **Example :** The relationship between class Student and Subject. One Student can register for many subjects and One Subject can enrol many students. Fig. 2.2.3(c) shows this relationship.

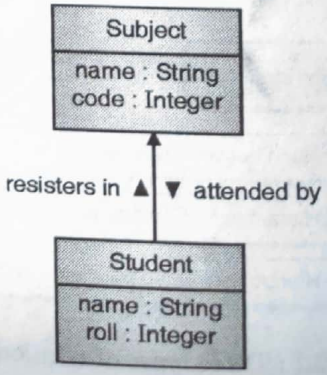


**Fig. 2.2.3(c) : Many to many association**

## 2.2.1(B) Class Diagram for Association between Multiple Classes

- The use cases of University application that have been illustrated in earlier sections, enables us to prepare a static model for the different sub systems.
- It will be easier task to identify the classes, attributes and operations if use cases have already been designed and explained. An arrangement of classes and their association is shown in Fig. 2.2.4.

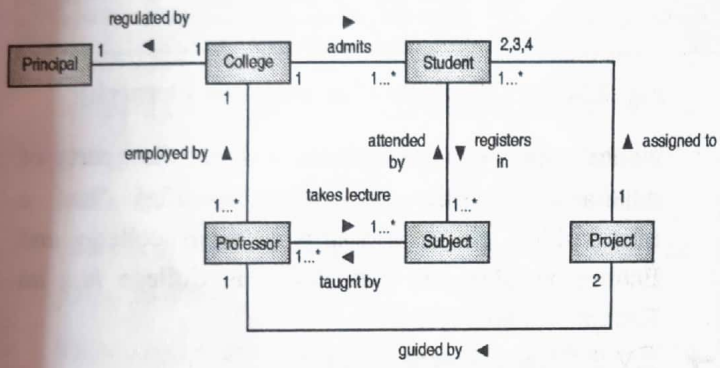


**Fig. 2.2.4 : Arrangement of classes and their association**

This class diagram shows association between different classes involved in one subsystem of University application. Multiplicity is also depicted in this class diagram. A brief description of each relation is provided below:

- There is **One-to-One association** between **Principal** and **College** class. A university can have many colleges and many principals. But one instance of college (a particular college) will have only one principal and one principal can regulate or administer only one college. So One-to-One relationship in both the direction.
- **One-to-Many association** between **Professor** and **College** class. As we know by the definition of One-to-Many association, there is One-to-One relationship in one direction and One-to-Many relationship in the opposite direction. So one Professor can belong to only one College (i.e. one to one relationship in one direction) but one college can employ many professors (i.e. one-to-many relationship in opposite direction)
- **One-to-Many association** between **Student** and **College** class. As we know by the definition of One-to-Many association, there is One-to-One relationship in one direction and One-to-Many relationship in the opposite direction. So one Student can belong to only one College (i.e. one to one relationship in one direction) but one college can admit many students (i.e. one-to-many relationship in opposite direction).
- **Many-to-Many association** between **Professor** and **Subject** class. By definition of many-to-many association, there is One-to-Many relationship in both the direction. So one instance of subject i.e. a particular subject is taught by many professors and in same way, in opposite direction of relationship, one professor can teach many subjects.
- **Many-to-Many association** between **Student** and **Subject** class. By definition of many-to-many association, there is One-to-Many relationship in both the direction. So one instance of subject i.e. a particular subject is registered by many students and in same way, in opposite direction of relationship, one student can get himself or herself enrolled in many subjects.
- **Numerically Specified association** between **Professor** and **Project** class: One special kind of associated that we left unexplored in last section is Numerically Specified Association which can be used to denote specific number of instances involved in relationship. For example, University has decided that one professor should guide exactly two projects, not more not less. Then it is necessary to specify such numbers in the relationship. So one professor can guide exactly 2 projects, that is shown in Fig. 2.2.4. Fig. 2.2.4 shows Project class box, denoting exactly 2 instances of projects involved in relationship. In opposite direction, one project can be guided by only 1 Professor.
- **Numerically Specified association** between **Student** and **Project** class: University has decided that one project can be assigned to only 2, 3 or 4 students. Then also it is necessary to specify such numbers in the relationship. So one project can be assigned to exactly 2, 3, or 4 students, is depicted in class diagram by mentioning these numbers below Student class, denoting only 2,3 or 4 students can work on one project. In opposite direction, there is simple one to one relationship i.e. one student can take only 1 project.

With this example we conclude our discussion on association of classes and in next section will discuss hierarchical relationship between the classes.

## ➜ 2.2.2 Class Hierarchies : Composition and Aggregation

➜ **(May 2017)**

**Q.** Explain aggregation, composition with reference to class diagram. **SPPU - May 2017, 3 Marks**

- In static modelling, complex real world object are mapped into abstract designing and programming world's objects. The object attributes and operation are first specified in suitable classes and relationships are identified.
- There are possible scenarios when identified class is made of other classes. Such relationship is described by the **composition** and **aggregation** relationship and also called **whole/part** relationship.
- Classes are connected by whole/part relationship and it is read as *Is part of.*
- Now take two examples to understand relationship of this kind and start thinking over it :
  1. A College class is composed of classes like Department, Admin Office, and Entrepreneur Cell. So it can be said that "Department *is part of* College or Entrepreneur Cell is part of College.
  2. We will discuss the example 1 in bit more detail, but first consider one more example of this kind. One Smart phone is composed of Battery, Screen, OS, RAM, Storage, and Processor. Here also it can be said that Processor *is part of* smart phone.
- Example 2 is an example of **composite (whole/part)** relationship. Object of class RAM, Processor, Screen, Storage are integral part of any instance of Smart Phone. There is one to one association between whole and its parts. One instance of Processor is part of exactly one instance of Smart Phone and removing the processor will make the phone incomplete.
- In the contrast of Example 2, Example 1 depicts a weaker relationship between the classes. First of all, there could be one-to-many association between the classes. One college, many departments. Removing one department will not shutdown the college. So better to avoid the term Whole/Part in this kind of relationship and use the term **aggregation**.
- In short, it can be stated that aggregation hierarchy is weaker version whole/part relationship. Fig. 2.2.5(a) shows composition hierarchy and uses the composite class Smart Phone and its parts to depict whole/part relationship.

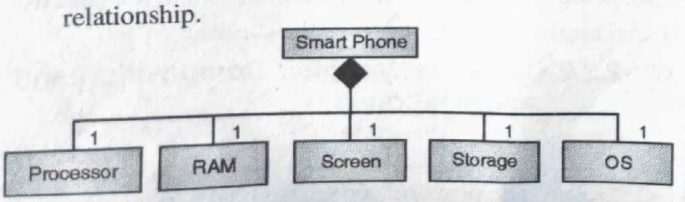


**Fig. 2.2.5(a) : Example of composition hierarchy**

- Fig. 2.2.5(b) shows the aggregation hierarchy and uses the aggregate class College to depict the aggregation relationship.

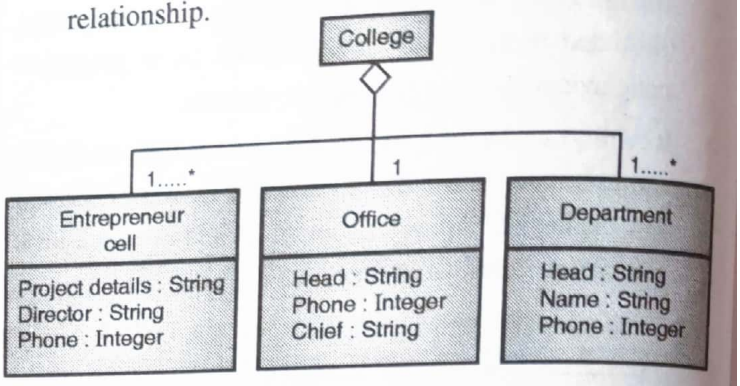


**Fig. 2.2.5(b) : Example of aggregation hierarchy**

- Notice that composition is called "Is part of relationship" while aggregation specifies "has a relationship". So relationship between college and Entrepreneurship cell can stated as College *has an* Entrepreneurship cell.

## ➜ 2.2.3 Class Hierarchies : Generalization and Specialization

➜ **(May 2017)**

**Q.** Explain generalization with reference to class diagram. **SPPU - May 2017. 2 Marks**

**Q.** Explain the concept of generalization and specialization of class hierarchies using suitable example and class diagram **(5 Marks)**

- Generalization and specialization is one of the interesting features of object orientation and every architect or designer must think how they can incorporate this concept in design to make the system simpler, easier and smaller.
- Generalization/specialization hierarchy defines the kind of parent and child relationship between the classes where child inherits attributes of the parent.
- In programming, the concept is often termed as Inheritance. A design with clean and clear modeling of generalization/specialization, helps the programmer to exploit the inheritance feature of programming languages.
- Inheritance has several advantages like code reusability, modularity and support for polymorphism (The concept will be discussed in later chapters).
- In **Generalization/specialization** relationship a parent class, called the super class has some generalized attributes and operations. A child class, called the sub class, is derived from parent class with inherited attributes/operations from parent plus its own attributes and operations.

- So, a super class represent a generalized class (common operations and attributes) and child represents specialized class (common + specific attributes and operations).
- Child class adds some unique attributes and operations of its own that makes it specialized and more detailed.
- The relationship between parent and child class is called "*Is a*" relationship.
- Generalization and specialization basically represent Inheritance (Child/Parent) relationship. The difference is just at the way of looking at the relationship. A class gets generalized (abstract) when we go upward in the hierarchy i.e. child to parent. A class gets specialized (detailed) when we go downwards in the hierarchy i.e. parent to child.
- There are many real life example of hierarchical relationship. Few are listed below :
  - Cow is an Animal
  - Car *is a* Vehicle
  - Teaching Staff is College Employee
  - Lecturer *is a* Teaching Staff
  - Assistant Professor is a Teaching staff

One can easily identify the relationship between these classes. Look at the last three examples. The most abstract class is **College Employee** with very abstract set of attributes like **employeeId** and abstract capability like **availStaffQuarters, getsSalary.**

A child class for **College Employee** is **Teaching Staff**. It has all the attributes and capabilities of its parent like **employeeId** and **availStaffQuarters,** additionally it has its unique attribute like **graduationDegree** and its own unique operation like **takesLecture.**

Similarly, child classes for **Teaching Staff** are **Lecturer** and **Assistant Professor**. They both have all the features (attributed and operations) of their parent, additionally they define their own.

Fig. 2.2.6 shows the relationship between the classes as described earlier.

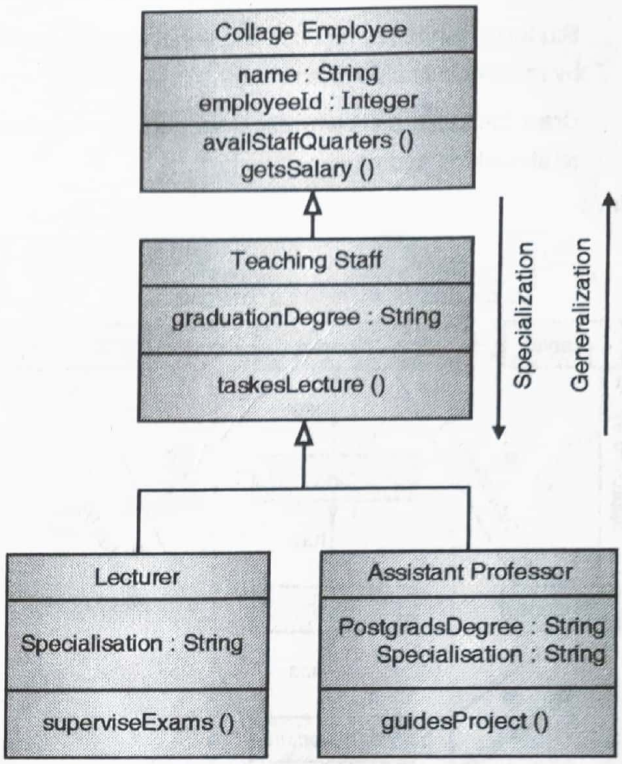


**Fig. 2.2.6 : Generalization/specialization hierarchy**

**Ex. 2.2.1**

Recently a well know publisher has launched their e-learning service for students. Users of the system will be system administrator, course creator and students. Administrator will manage the system and start the initial system setup and configuration. Creator will create the video lectures for the courses assigned by administrator. Students can register, login download and subscribe the video lectures by paying online.

**Following is the work flow of the system :**

1. Administrator creates and configures all the courses (subjects)
2. Subject can have multiple units.
3. Units can have multiple topics.
4. Each topic has exactly one video lecture.
5. Administrator assigns each course to one or more creators.
6. Creators can record video lecture and upload in the system

7. Creator can only upload for assigned courses.
8. Students can login, register and subscribe the course by making online payment.

Draw the class diagram with clear depiction of above relationships and multiplicity.

**Soln. :**

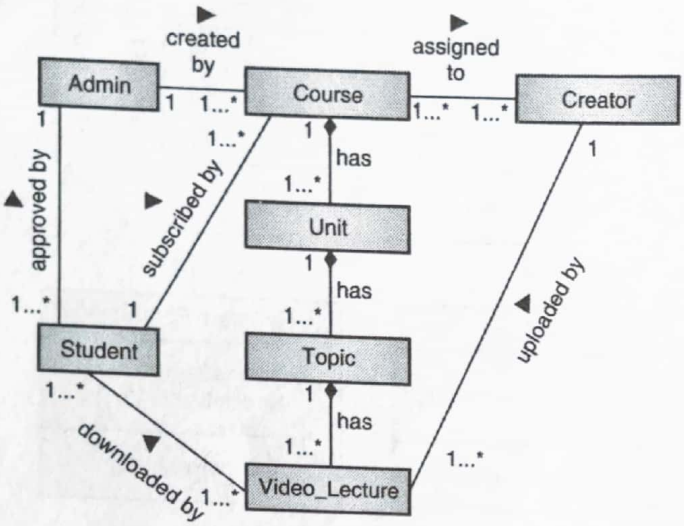


**Fig. P. 2.2.1(a)**

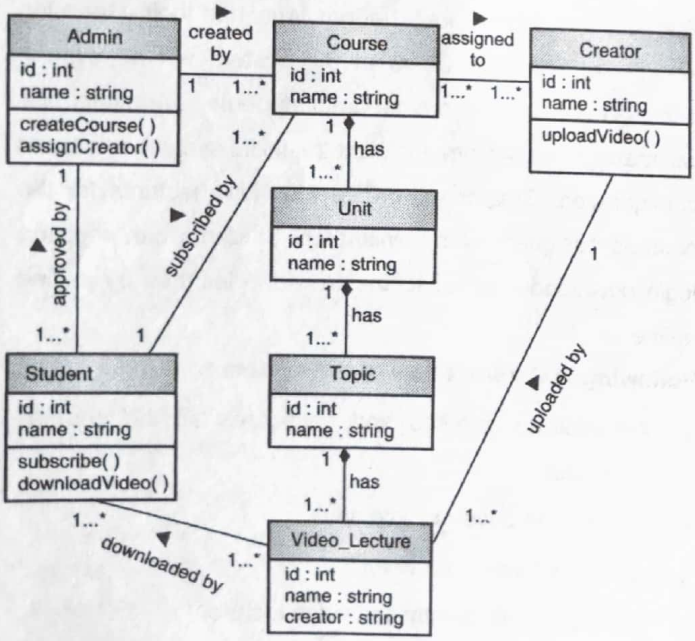


**Fig. P. 2.2.1(b)**

**Ex. 2.2.2** **SPPU - Feb. 2016 (In sem), 5 Marks**

Draw a state machine diagram for coffee vending machine.

**Soln. :**

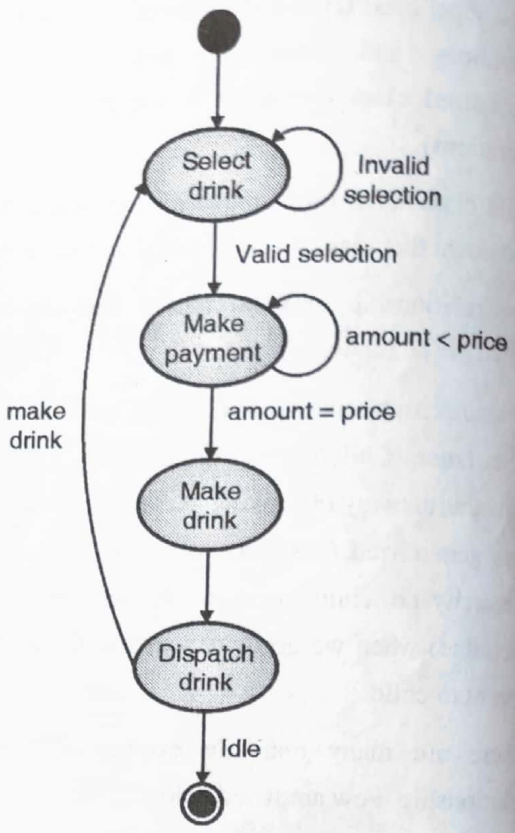


**Fig. P .2.2.2**

**Ex. 2.2.3**

Draw the class diagram for online shopping system with following assumptions.

1. System is web based application providing the services to buyers and sellers.
2. Buyers and seller can create account.
3. Seller is responsible for fulfilling the orders.
4. Buyers creates order and make payments.
5. Currier is responsible for delivering the parcel.
6. Order is created once the payment is done.

**Soln. :**

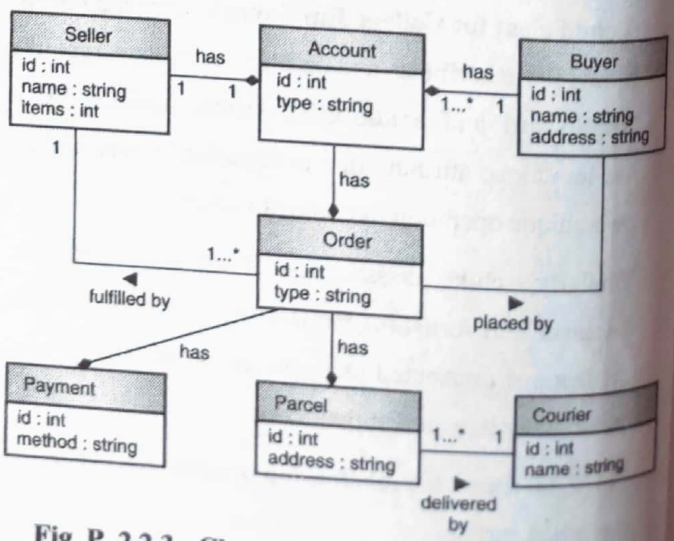


**Fig. P. 2.2.3 : Class diagram for e-shopping system**

**Ex. 2.2.4** SPPU - Feb. 2016(In sem), 5 Marks

Draw a sequence diagram for online shopping system.

**Soln. :**

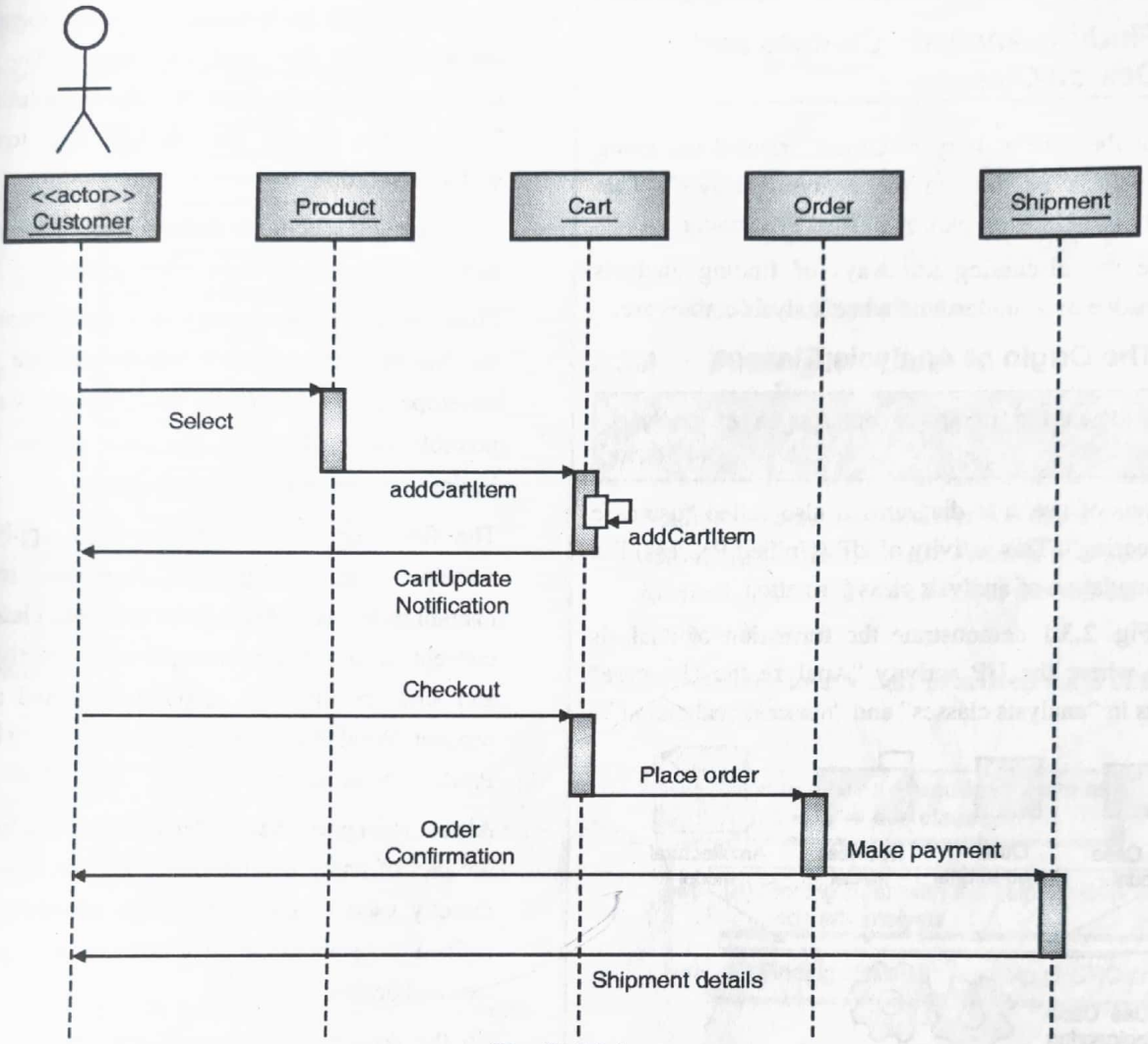


**Fig. P. 2.2.4**

## 2.2.4 Difference between Class Diagram and Object Diagram

➔ **(May 2016)**

**Q.** How is class diagram different from an object diagram? SPPU - May 2016, 3 Marks

| Parameters | Class Diagram | Object Diagram |
|---|---|---|
| Definition | Class diagram captures the static aspect of system to be designed. It shown static relationship. | Object diagram depicts behavioral relationship between class instances at a point in time. |
| Dynamic change | It does not include dynamic change in the system. | It captures the dynamic or runtime changes in the state of system. |

| Parameters | Class Diagram | Object Diagram |
|---|---|---|
| Data | It never includes specific data values of entities or attributes in the system | It may include data values of entities or attributes in the system |
| Purpose | Class diagram shows what objects in system consists. | Object diagram shows how those object behave at run time |

**Syllabus Topic : Classes Finding Analysis and Design Classes**

## 2.3 Finding Analysis Classes and Design Classes

- One of the core activity of Object oriented designing and analysis is "finding the analysis classes". This activity is an integral part of Unified Processing.
- Before the discussing the ways of finding analysis classes, we must understand what analysis classes are.

### 2.3.1 The Origin of Analysis Classes

**Q. Write down the origin or emergence of analysis classes. (4 Marks)**

- Analysis of use case diagrams is also called "use case engineering". This activity of UP (Unified Process) lies the foundation of analysis class formation.
- The Fig. 2.3.1 demonstrate the formation of analysis class, where the UP activity "Analyze the Use case" results in "analysis classes" and "use case realization".

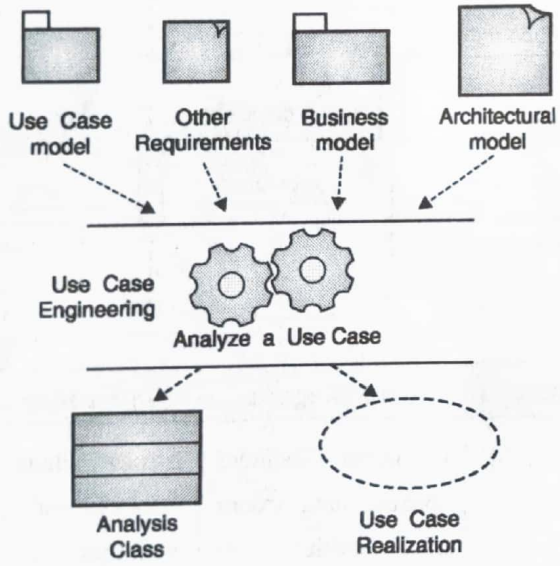


**Fig. 2.3.1: The Origin of Analysis Class**

- The process use case engineering carries out the analysis of use case that result into use case realization and analysis classes. In this section we will be learning the analysis classes in detail.

### 2.3.2 Defining the Analysis Classes

**Q. How to define the analysis class? (2 Marks)**

- Analysis classes can be defined as :
  - The classes that represent highest level of abstraction in the problem domain
  - The classes which has direct mapping with actual real world business concepts.
- The real world business concepts are formulated in software using the analysis classes. The problem domain is where the need of software solution arises. The analysis classes are the first step towards the software solution.
- These are too specific to the real world busses concepts like "product sell, package delivery, etc.
- Fundamental requirement by an analysis class is to map the business concept (for which software is getting developed) in a completely unambiguous way. This is possible only when the use case models and other business model remains unambiguous.
- The first step to build software by applying OOP principles. It makes the business requirement unambiguous and clean. It includes all clear business concept definition. For example what exactly a product and who are involved in purchasing and selling the product. What attributes a Customer have, what exactly Product Order mean etc.
- Also it is important to understand that analysis classes are part of analysis modeling. Analysis classes are not directly used in software design modeling. They are refined or elaborated or transformed to be used in detailed design.
- So the design considerations are not associated with analysis classes. Hence the analysis classes are associated more with problem rather than solution domain.

### 2.3.3 Anatomy of Analysis Classes and Design Classes

**Q. Write down the anatomy (components) analysis class and design class. (2 Marks)**

- Analysis classes are responsible for highlighting high level of abstract attributes. This gives the complete ides of business concept and related attributes.
- The design classes are further refinement over analysis classes or sometimes elaboration of analysis classes. Analysis classes provides foundation for design classes.
- The high level services and attributes are specified in analysis class. These services takes the form methods or functions in design classes. These services are implementable methods in design classes.

- An analysis class uses small subset of design classes in terms of UML notations. Implementation details are intentionally neglected in analysis classes.

☞ **General components of analysis classes**

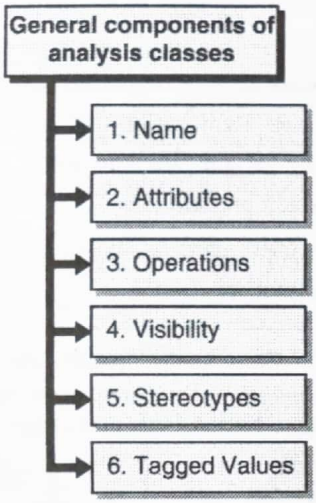


**Fig. C2.4 : General components of analysis classes**

➜ **1. Name**

Name of the class is always mandatory.

➜ **2. Attributes**

The properties of the identified business concept is mentioned as attribute. Attribute types are not needed to be specified.

➜ **3. Operations**

Services are specified analysis class as operations. Later on in design classes, these operations get converted into methods or functions.

➜ **4. Visibility**

Normally visibility are shown in design not in analysis classes. But sometimes if needed they can be shown in analysis classes two.

➜ **5. Stereotypes**

Though these are integral to the design classes, can be shown in analysis classes also if it enhances the clarity of model.

➜ **6. Tagged Values**

Can be shown to increase the clarity of model.

☞ **Example**

- For a library management system the concept class or analysis class Book can be shown in Fig. 2.3.2.

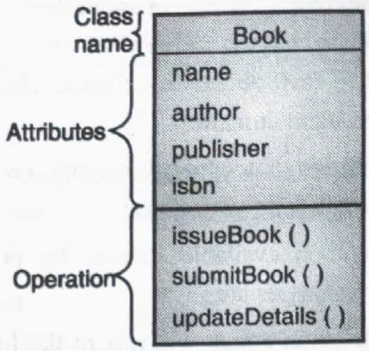


**Fig. 2.3.2 : Example of analysis class book**

## 2.3.4 Finding the Classes

**Q. Write down the ways of finding the analysis classes. (2 Marks)**

- Finding the analysis classes is one of the first important activity in Unified Process. There is no fixed algorithm for finding the analysis class but the standard practices are there in place.

☞ **Well-defined and widely practiced ways of finding analysis classes**

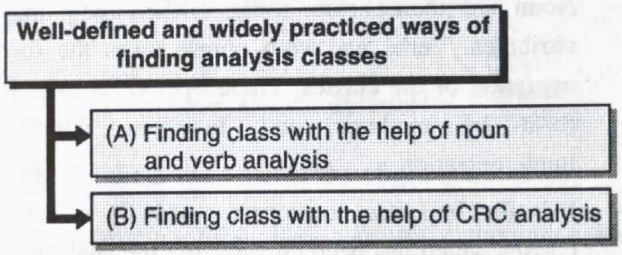


**Fig. C2.5 : Well-defined and widely practiced ways of finding analysis classes**

### ➜ 2.3.4(A) Finding Class with the Help of Noun and Verb Analysis

**Q. How to find analysis class using noun-verb analysis ? (4 Marks)**

- This is the text analysis of the requirement where more and more noun and verbs are analyzed to be identified as classes.
- For example in Banking System application, Customer, Transaction, Account, Employees, Deposit, Withdraw are the noun/verb which can be converted into analysis classes.
- Sometimes the verb like 'Deposit and Withdraw' can be identified as operations within the analysis classes. During the detailed design phase however theses may be converted into design classes.

- One important aspect of noun/verb analysis is to identify the hidden classes which are generally not mentioned in problem description. These are intrinsic to the problem domain.

The analysis procedure for noun and verb can be formulated in the following steps :

- The common available sources for performing noun and verb analysis are :
  1. The requirement analysis in the form of text and stories or
  2. Use case model and stories
  3. Project description document or Glossary
  4. Other informal documents
- After availing the above sources, following aspects are highlighted for noun/verb analysis
  1. Noun (e.g. customer, employee)
  2. Noun phrase (Bank account)
  3. Verbs (deposit, withdraw)
  4. Verb Phrases (Verify User Credentials)
- Noun and noun phrases serves as the class names and attributes. Verbs and verb phrase takes the form of operation of the classes. These operations exist in the form of methods and functions during the implementation.
- It is also important to prepare a wide list of candidate classes which has been encountered the analysis. Later on they can be filtered to prepare the final list of analysis classes.
- Assignment of operations and attributed to the list potential candidate classes can be done using the CASE (Computer Aided Software Engineering) tool.
- It is also possible during the analysis that relationship is identified between the classes. This serves as candidate relationships and is of important use during the design phase. Relationship between design classes is crucial for system implementation.

## ➜ 2.3.4(B) Finding Class with the Help of CRC Analysis

**Q. Explain finding the analysis class using CRC analysis. (4 Marks)**

- Another way of finding analysis classes is the CRC which is considered very interesting and effective. CRC stands for class, responsibility and collaborators.
- The session of CRC is brainstorming session where sticky notes are used to identify and list important things in the problem domain.
- The session can be attended by analysts, architects, developers, users and other stakeholders of the proposed system.
- Sticky notes are divided into three sections and marked as Class Name, Responsibilities and Collaborators at the beginning of the session and distributed among different session participants.
- The sample CRC card is shown in Fig. 2.3.3 where three section marked.

| Class name: Registration | |
|---|---|
| Responsibilities:<br>Create Patient Record<br>Generate Bill<br>Generate Receipt | Collaborators:<br>Patient<br>Doctor<br>Ward |

**Fig. 2.3.3 : CRC card sample**

- Fig. 2.3.3 shows that CRC card is for hospital management systems for patient's admission section.
- The first compartment at top is the name of candidate class. The left is reserved for responsibilities to be identified and right one for collaborators.
- The collaborators are other classes in the systems that interact with this candidate class or are involved in the operation of this candidate class. Hence the collaborator section is responsible for identifying relationship between the classes.

☞ **Two phases in CRC analysis**

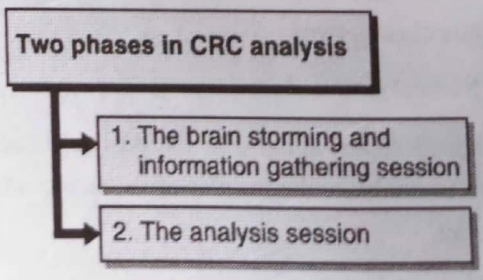


**Fig. C2.6 : Two phases in CRC analysis**

➜ **1. The brain storming and information gathering session**

This is for collecting the information by having discussion or self-analysis by different stakeholders and domain experts.

**2. The analysis session**

Once the raw information is gathered in phase 1, now is the time to categorize it to place in the appropriate compartment in CRC card. Here it is decided that which information will become class and which one will become attribute.

**Syllabus Topic : Refining Analysis Relationships**

## 2.4 Refining Analysis Relationship

**Q. Explain the refinement of analysis relationship. (5 Marks)**

- Analysis classes also highlights the relationship that is going to be integral part of object diagram of design phase.
- Relationships may exist in the form analysis class during the initial phase of "finding analysis class" activity. Sometimes the relationship is derived from Collaborators in CRC.
- Refinement of the relationship is done at the time of formulating design class and at the time of formulating object diagram.
- Relationship between the classes exist in the following form :
  1. Association between the classes (Refer Section 2.2.1)
     a. One-to-One Association
     b. One-to-Many Association
     c. Many-to-many Association
  2. Hierarchical relationship
     a. Composition and Aggregation (Refer Section 2.2.2)
     b. Specialization and Generalization (Refer Section 2.2.3)
- All the above mentioned types of relationship has been cover in previous sections. Please refer section 2.2.1, 2.2.2 and 2.2.3.

**Syllabus Topic : Inheritance and Polymorphism**

## 2.5 Inheritance and Polymorphism

- Inheritance and Polymorphism are the two most used principle and design practices in modern application development. They are two of the four pillars of object oriented programming.

**Definition of inheritance and polymorphism**

**Q. Define inheritance and polymorphism. (4 Marks)**

### Inheritance

- Inheritance encourages the concept of reusability where effort in designing and coding can be saved with convenience.

### Polymorphism

- Polymorphism supports that multiple complex operations can performed in simplest possible way. It increases code and design extendibility, manageability and cleanliness.
- Inheritance is based on the concept of generalization and specialization discussed in section 2.2.3. The Fig. 2.5.1 the generalization hierarchy.

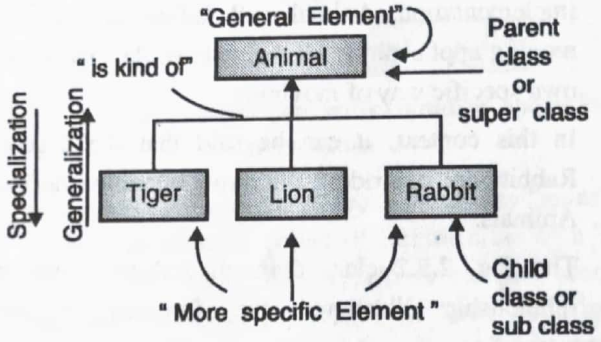


**Fig. 2.5.1 : A class hierarchy of generalization.**

- Let us discuss these two concepts (inheritance and polymorphism) in detail and their applicability in design.

### 2.5.1 Class Inheritance

**Q. Describe class inheritance. (2 Marks)**

- The generalization hierarchy implicitly causes the classes to be in inheritance relationship. The subclass inherits all the features of superclass.
- In general, the subclass inherits the attributes, operations, relationships and other constrains from the super class.
- Also in most cases, subclass adds new feature of its own, which is not present in superclass.
- It is possible too that subclass override the superclass operation. It means the subclass will have the same operation but the behavior will be different. This concept is called overriding and helpful to implement polymorphism.
- In next few sections, we will discuss the various aspects of class inheritance.

## 2.5.1(A) Overriding

**Q. Explain overriding in relation with inheritance. (4 Marks)**

- The concept overriding is associated with inheritance. When a superclass defines a function or method or operation and the same function or operation is again defined in subclass with new behaviors (subclass specific), the phenomenon is called overriding.
- Consider a superclass Animal which has method move( ). The subclass Tiger has move( ) method which will cause Tiger to move differently than generic behavior of Animal's move. Again the move( ) method in Rabbit class will have different implementation.
- So all three classes Animal, Tiger and Rabbit has the move( ) operation but they may have their different implementation. Animal will define generic way of moving applicable to all Animals while Tigers have its own specific way of moving.
- In this context, it can be said that the Tigers and Rabbits are overriding the move operation defined by Animals.
- The Fig. 2.5.2 class diagram and their inheritance relationship illustrated the function or method overriding.

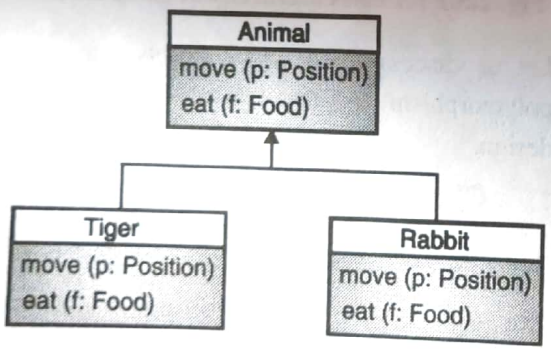


**Fig. 2.5.2 : Overriding**

- It should be noted that overriding is considered valid only if functions or methods have exactly same signature in subclass and superclass. Signature of operations mean name, return type, type and number of parameters.

## 2.5.1(B) Abstract Operations in Classes

**Q. Explain abstract operations of classes. (6 Marks)**

- In many cases of inheritance, the implementations of operation (functions or methods) are deferred to subclasses. Such operations which do not have any implementation logic inside it is called abstract operations.
- The sole purpose of abstract operations in a superclass is to define an operation that must be implemented by its subclass. In general such operation give the idea of the capability of all subclasses of a superclass.
- The Animal class may define its operation move( ) as abstract operation. Doing so, makes it mandatory for all of Animal's subclass to provide implementation of move( ) operation.
- It also logically makes sense that we should leave move( ) method in Animal class empty as the generic behavior for move( ) cannot be defined for generic class Animal. We are quite sure that we are never going to create direct object of Animal class.
- In many programming languages, defining the operation as abstract makes the class abstract, which in turn restricts user from creating object of that superclass. These benefits in programming by restricting the accidental creation of superclass (abstract) object.
- The example that we have discussed in previous section, the Animal class has move( ) operation, which should ideally be defined as abstract, that will make out Animal class an abstract class. So we will never be able to create direct object of Animal class.
- Any class extending the Animal class, is basically getting engaged in a contract that class must implement the move( ) and eat( ) operations. Now the way Tiger will move (say in an android gaming app) will be different from the way Rabbit moves. But we should never see an animal moving in our gaming app. Because the concept Animal is too abstract to visualize in game and we rightly declared it an abstract class.
- Fig. 2.5.3 shows the inheritance relationship between the Animal and subclasses.
- Here we have declared move( ) and eat( ) as abstract operations that in turn makes the class Animal an abstract class. Any class extending Animal class must provide implementation both the abstract operation.

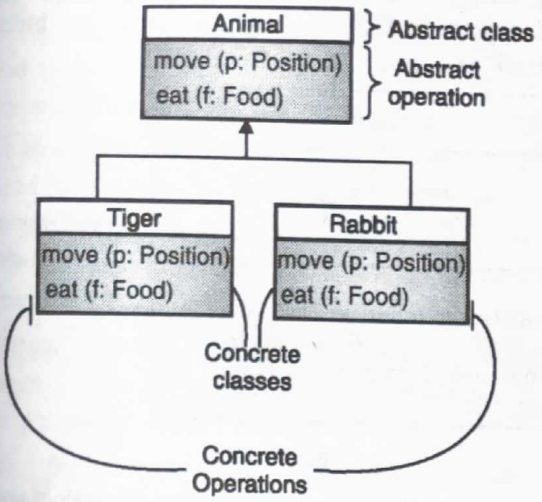


**Fig. 2.5.3 : Abstract class operations**

## 2.5.2 Polymorphism

**Q. What is polymorphism? Explain with example.(5 Marks)**

- The meaning polymorphism is "many forms". In context of Object orientation, the same operation has many implementations.
- Programming languages provide more specific definition of polymorphism according to language capability and support. For example run time and compile time polymorphism.
- We can visualize the polymorphic behavior among different king of Animal objects. move( ) operation is taking different forms in Tiger and Rabbit object. Both will move differently on screen. So the same operation has different implementation in different classes.

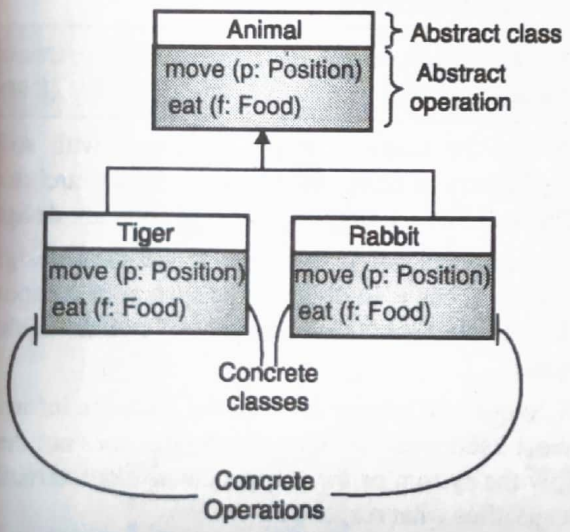


**Fig. 2.5.4 : Polymorphism in classes**

- Polymorphism is of great use to make the design and code simpler and disciplines. The move( ) operation will be invoked on an Animal class handle (a reference of Animal type) and it will depend on the context which implementation of move( ) will get called.
- Fig. 2.5.4 illustrates the concept of polymorphism with same example of Animal superclass and its subclasses.
- We can further elaborate the design to show the context based runtime polymorphism.
- In the Fig. 2.5.5, it is shown that a reference or a variable of type Animal can be used to hold the object of any of its subclass.

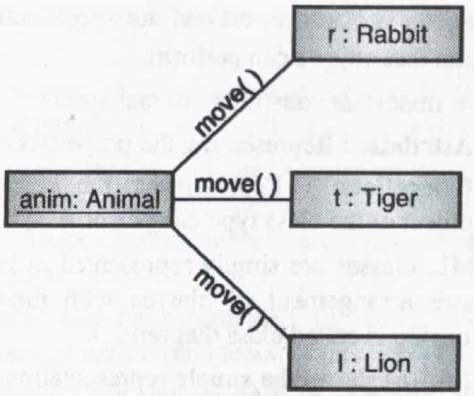


**Fig. 2.5.5 : Polymorphism with Dynamic operation invocation**

- Same reference variable of type Animal say "anim" can be used to store an object of Rabbit class or a Tiger class. It depend on the context or runtime scenario. Now calling move( ) or eat( ) on "anim" can invoke Tiger's specific operation or Rabbit specific operation, depending on the type of object "anim" is referring to.

**Syllabus Topic : Object Diagram**

## 2.6 Object Diagram

**Q. Explain the concept of object diagram. (6 Marks)**

- The objects in object diagram are identified in problem domain keeping the use cases in mind. The objects are identified with its properties and capabilities. The actions of capabilities are not taken into consideration however in static modeling. Classes are used to declare the properties and behaviour of the object.
- Class represents the abstraction of real world object.
- Objects of similar kind belong to one class. Collection of objects of similar characteristics is class. It captures static structure of the system.
- In programming languages, class is prototype or blueprint using which objects are created. We can treat classes as user defined data type.
- In designing, the concept are mapped in objects and objects are statically represented as classes. For

example Student is class. Subject is a class. Objects are nothing but instances of the class.

- Objects on the other hand, is not a concept or declaration but an existing "thing" that posses some characteristic and behaviour. Object belongs to some class that is already declared and understood.
- The properties or attributes held by object is represented in class as variables.
- Behaviour of object is represented in class as operations or methods.
- Operations declared in classes are specification of the function that objects can perform.
- So two important constituent of classes are:
  1. **Attribute :** Representing the properties of object
  2. **Operations :** Representing the functions that object of the class type can perform
- In UML, classes are simply represented as boxes. One or more arrangement of classes with their possible relationship is called class diagram.
- Fig. 2.6.1(a) shows the simple representation of classes and objects in UML.

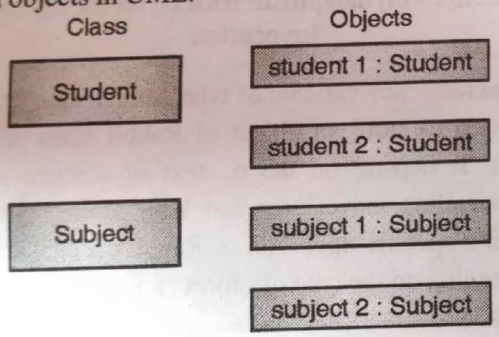


**Fig. 2.6.1(a)**

- Fig. 2.6.1(b) shows the simple representation of classes and objects in UML with properties or attributes.

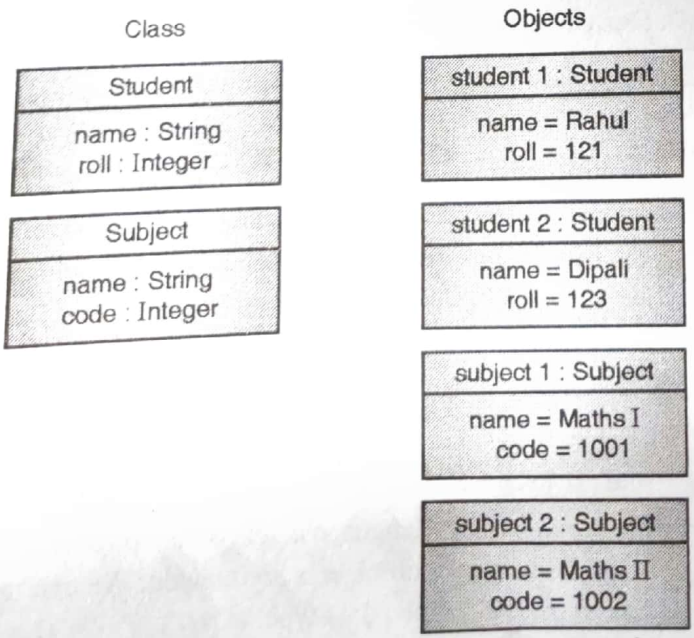


**Fig. 2.6.1(b)**

- Fig. 2.6.1(c) shows the simple representation of classes and objects in UML with properties or attributes and operations.

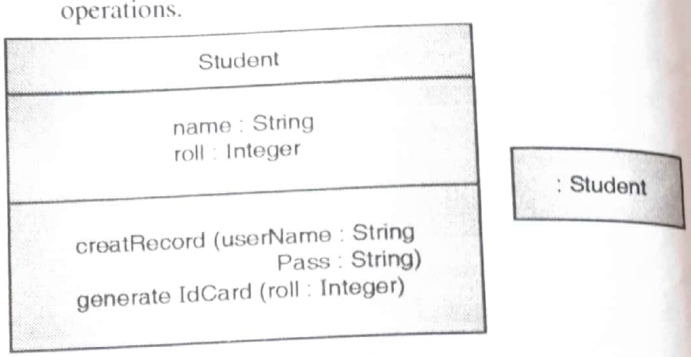


**Fig. 2.6.1(c)**

- Operations reflect the capabilities of the object and task that it can perform.
- The complex association between the classes can be captured in a dynamic representation object and their interaction in an **object diagram**.

**Syllabus Topic : Component Diagram : Interfaces and Components**

## 2.7 Component Diagram : Interfaces and Components

- One of the major way of capturing the structural aspect of software is component diagram. The system is described in the form connected components.
- Interfaces and components are the basic building block of component diagram. Let us discus these elements in component based software architecture.

### 2.7.1 Component Based Software Architecture

**Q. Describe component based software architecture in brief. (6 Marks)**

- One of the major concepts associated with software architecture is component based software architecture. This is the structural perspective of software design.
- In component based software architectural design, the primary focus is in dividing the system in components. Each component is well defined self contained block of system.
- A component encapsulates all the complex information about itself inside it. So, a component does not describe how the system or the component works internally but it specifies what responsibility it has.
- A component can be a simple object like Student object or it can be a composite object like Department object containing other simple objects.
- A component also specifies the interface through which it communicates with other components.

- One component does not need to know all the information about the other to communicate. It just need to know what methods are offered by the other component's interface to communicate.
- For example **Student** object may offer an interface called the **Interaction_Interface.** It offers two methods say **getStundentRecord** and **setStuendetRecord.** So other components just need to know these two methods to communicate with Student object.
- A component if often considered as black, the internal design description and implementation of that component is hidden from other component.
- One component can communicate with other components by several possible ways by applying suitable communication pattern.
- There are several communication patterns which can be applied while designing the communication. Patterns are nothing but well practised design strategy applied on similar kind of problems.
- We explore few architectural and communication pattern in subsequent section of this chapter. The major design patterns will be covered in next chapter.
- A component design strategy is often called the sequential design strategy where components are represented by class and component instances are represented by objects.
- Components are so independent and isolated that it often considered as plug-and-play while designing. It can assumed at the initial stage that certain kind of component can be compiled separately and kept in library of components. When the functionality is needed, that particular can be attached to the system.
- In distributed environment or networked application, components are active entities which can be deployed at any network node. For example there could be several components representing the clients and one component representing the server.

## 2.7.2 Component Diagram

**Q. Describe component diagram with example. (5 Marks)**

- Component diagram divides the entire system in different connected components. Here the component diagram captures the structural view of software system.
- The structural view of software system is the static view and highlights the overall architecture of system.
- It represents the overall architecture in the form of different organized subsystems connected with each other. These subsystems are represented as components.
- This view can be represented by UML subsystem component diagrams and associations between them.
- The structural view covers only those aspects which do not change with time. It is the most stable view and all subsequent views are usually designed based on this static view.
- Composite and aggregate classes are used to design the subsystems and their relationship can be depicted by associations and their multiplicity.
- Now we will carry on our discussion based on the case study specified in earlier section.
- A structured view of the medical centre application is shown in Fig. 2.7.1. A class diagram is used to depict the subsystems.

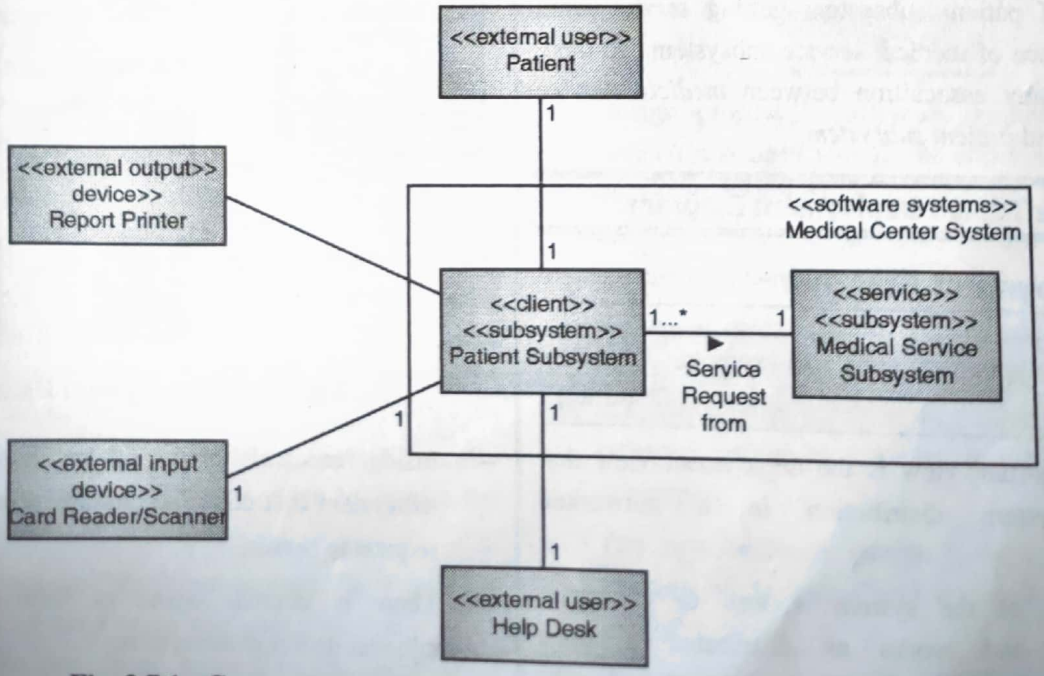


**Fig. 2.7.1 : Component Diagram for Structural View of Software System**

- The medical centre system is divided into a patient subsystem and a medical service subsystem. Stereotypes <<client>> <<service>> are used to characterize the system.
- Patient subsystem is a light-weighted client contains the minimal functionality and accessed directly by the user. The role of user can be played by patient, doctor, pathologist etc.
- Patient subsystem request for the services from medical service subsystem that contains the core medical functionality and patient's central database repository.
- Many patients can access the medical centre application in parallel. A user can swipe the card and check the personal information through the client system installed in hospital. Another at the same can access the client subsystem online and doctor can write the prescription patient. So there are multiple instances of client sub-system or the patient subsystem.
- There is single instance of *medical service subsystem* that contains all the data processing capabilities and data repository.
- In one use case scenario, a patient can register himself using the patient subsystem. The patient's information is temporarily stored in client sub system. Once approved by system administrator, it is ultimately submitted to medical service sub-system.
- The structured view of software architecture also specifies the associations and multiplicity among the subsystems. For example, there could be multiple instances of patient subsystem getting service from single instance of medical service subsystem. So there is one-to-many association between *medical service subsystem* and *patient subsystem.*

**Syllabus Topic : Deployment Diagram**

## 2.8 Deployment Diagram

**Q.** Describe deployment diagram with example. **(7 Marks)**

- Another important view is the deployment view that captures system distribution in a networked environment.
- Today most of the system is web or enterprise applications and works as distributed systems. Deployment view specifies the location of each sub-system and their communication media.
- Deployment view can be depicted as deployment diagrams. It shows the deployment of subsystems at different network nodes and the type of network.
- Deployment views are more concerned about physical layout of software system. The idea is to capture the scope of system in terms of installation or deployment.
- In general it is straight forward block diagrams of subsystems and their communication media. However architects can choose to add distributed configuration details in it. For example: type of communication media, interfaces to the subsystems and protocols being used etc.
- Deployment diagram for medical centre application is shown in Fig. 2.8.1. Here several client subsystems are possible which can be deployed on different nodes within hospital premises and laboratory.
- The Fig. 2.8.1 shows the sub systems of medical centre application. There is one instance of server called the medical service subsystem deployed at central server called the server node.
- There are several instances of clients deployed at different client nodes. For example the web portal is hosted at one client node that can be accesses remotely by the patients through Internet.

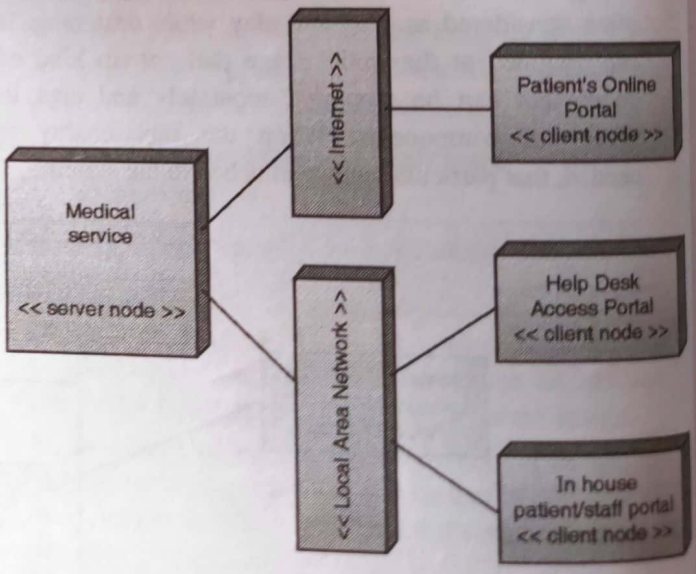


**Fig. 2.8.1 : Deployment Diagram**

- Inside medical centre, there is dedicated client subsystem that can read patients access card and send request to server.
- There is internal layout of local area network to connect different subsystems.

**Syllabus Topic : Package Diagram**

## 2.9 Package Diagram

- Packages in software are used to group logically similar kind of elements.
- In coding, packages are used to group similar classes. Similarly in designing, similar elements of design are grouped into design packages.
- UML specifies the packages as standard design elements.

### 2.9.1 Packages in UML

**Q. Write use of package. (2 Marks)**

- UML encourages the grouping of similar elements into a package. Package serves as container and owner for model elements.
- Often the package is termed as UML mechanism for grouping the things of similar types.
- Package is also to collect the diagrams in groups. Package can be used to:
  - Collect similar kind of elements (semantically) into one container.
  - Define the semantic boundary of the model
  - Clear logical understanding of model and to manage complex designs
  - Provide a namespace for similar kind of elements. Element names can be repeated among different packages but it must be unique inside a package.

### 2.9.2 Use Case Packages

**Q. Why it is good practice to group use cases in a use case package? Explain the package diagram with and UML notation. (6 Marks)**

- A large application software project like University Application can have large number of possible use case scenarios. Different subsystems like Attendance, Admission, Leaves, Examination has different set of use case scenarios.
- **Use case packages** are used for grouping logically similar kind of use cases at one place to handle the complexity.
- A use case package therefore represents the high level functionality of the system and contains a number of more specific use cases related to the high level functionality.
- Sometimes use case packages are formed base on the actor that interacts with system. So it is possible to have a dedicated package for Student's uses cases and so for others.
- Fig. 2.9.1 **shows** the Use Case package for Student_Id_Card subsystem. The name of package is "StudentRecordManagement" and the primary actor for this package is Student. Other actor is Application Administrator.

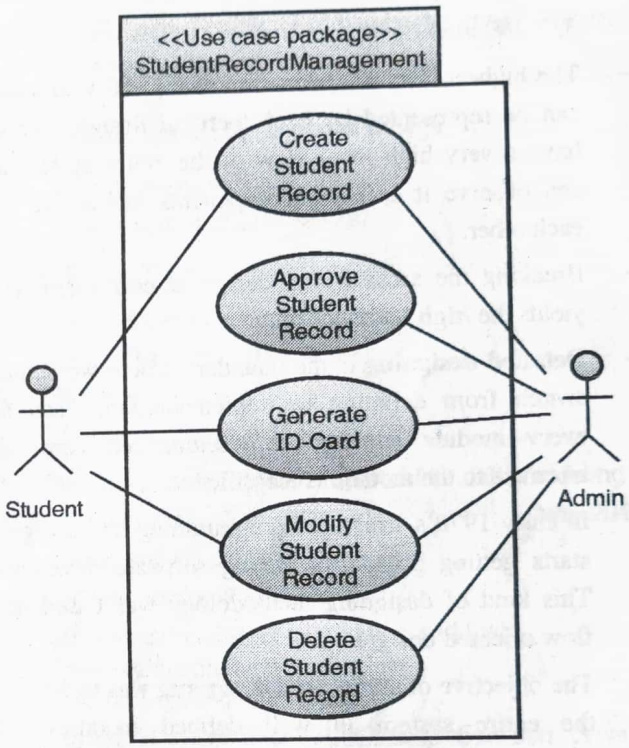


**Fig. 2.9.1 : Package Diagram for Use case**

**Summary**

- Design process in software projects are very well defined and need significant effort before starting the actual implementation or coding. The approach of design is often termed as "**Model based Software Design and Development**".
- **Software design** is often considered as initial steps towards the solution with available capabilities. There could be multiple designs for the same software system covering different aspects of the system to be developed.
- The term **Software design** is used to refer both, the **process** of designing and the design product or the **model**.
- **Design process** is where the creativity lies. An experienced and skilled architect needs to be creative

also in the modern days of competitive software development strategies.

- A **model** is nothing but the architecture that describes all aspects of the system that we want to build.
- The outcome of software design is often classified in three levels :
  1. Architectural Design
  2. High Level Design
  3. Detailed Design
- The highest level of abstraction of the software system can be represented by **architectural design**. We can have a very high level view of the entire system and can observe it as many components interacting with each other.
- Breaking the architectural design at component level yields the **high level design.**
- **Detailed designing** is the boundary where we actually switch from deigning to implementation. Each and every module structure is implemented here. The interface to the module is identified.
- In early 1970's structured programming and designing starts getting popularity among software developers. This kind of designing methodology was based data-flow oriented designing.
- The objective of **structured designing** was to organize the entire system in well defined modules. The structured designing was one of the first well documented designing methodologies.
- Two important approaches of designing were introduced with structured designing. They were called coupling and cohesion, two important aspects of modularity.
- **Procedural design** is even more conventional design methodology where design decisions are made even when the system functionality is not understood properly or even when all the information is not available.
- One of the most popular design methodology widely practised from desktop, server to even embedded systems is **Object oriented design and analysis**.
- **Object orientation** is way of thinking a problem with real world perspective. The concept of the system in mind is mapped in the objects, their properties behaviour and interaction.
- **Object orientation** avoids the problems and pitfalls associated with conventional designing methodology.
- The concepts that we mostly address in **object orientation** approach is classes, objects, inheritance, generalization, specialization, polymorphism and abstraction.
- **Objects** are the central concept in object orientation. It represents the abstraction of any real object in problem domain. It reflects the capability of the system. It reflects what information system will keep and also it reflects what methods it offers to interact with the system.
- Information hiding or **abstraction** is another central approach of object oriented designing. If a module is well contained with its own information, it will be less dependent on other modules and hence changes can be easily done without affecting the other modules.
- **Object orientation analysis** is the first method applied in object oriented methodology of software development. This is the analysis phase of software development. The mapping of real world to programming world is performed here. Objects are identified along with their properties and behaviour. The phase is often called **object modeling**.
- **UML** (Unified modeling Language) is standard modeling language and notation for object oriented analysis and design.
- **OMG** (Object management group) has adopted UML as standard notation for designing in 1997. The most significant contributors in forming the notations and specifications of UML are Grady Booch, Ivar Jacobson and James Rumbaugh. The design and notation of UML was developed by them at Rational Software.
- **Use cases** capture the functionality of the system from user's perspective. It is represented as the sequence of interaction between **actor** and the **system.**
- Use case diagram was proposed by Jacobson in the book subtitled *"A use case driven approach"* 1992.
- Use cases are described in texts as well in diagrams. Text form of use cases are called ***use case descriptions.***
- **Activity diagrams** are used to represent flow of different activities in the system or sub system.
- **Activity diagrams** are standard UML diagrams and have well defined notations.
- **Activity diagrams** shows the flow of control and sequencing among different possible activities.

- **Activity diagrams** do not explore system design or system's processing logic, instead focuses on flow of activities that a user of the system can experience.
- In static modeling, we capture static view of the system. Structure of the system that does not change with time is modeled in static modeling. The concepts of problem in hand are translated into designing constructs called **classes.**
- In programming languages, **class** is prototype or blueprint using which objects are created. We can treat classes as user defined data type.
- Two important constituent of classes are:
  1. Attribute : Representing the properties of object
  2. Operations : Representing the functions that object of the class type can perform
- The static relationship between the classes is depicted by associations.
- One important aspect of association is multiplicity that defines how many instances of one class is associated with single instance of another class.
- There are possible scenarios when identified class is made of other classes. Such relationship is described by the composition and aggregation relationship and also called whole/part relationship.
- Generalization/specialization hierarchy defines the kind of parent and child relationship between the classes where child inherits attributes of the parent.
- In programming, the concept is often termed as Inheritance. A design with clean and clear modeling of generalization/specialization, helps the programmer to exploit the inheritance feature of programming languages.
- Inheritance has several advantages like code reusability, modularity and support for polymorphism (The concept will be discussed in later chapters)
- In **Generalization/specialization** relationship a parent class, called the super class has some generalized attributes and operations. A child class, called the sub class, is derived from parent class with inherited attributes/operations from parent plus its own attributes and operations.
- **State** of the system can be represented by the collection of individual states of objects that form the system.
- In programming, **object states** are nothing but the values of the attributes at any given instance of time. For a dynamic system, its object's state keeps on changing as objects interacts with other objects.
- **Sequence diagrams** are used to show to interaction between objects when system becomes operational.
- The sequence of interaction in **sequence diagram** is decided by messages passed between objects.
- Dynamic view of the system is captured using the **sequence diagram**.

## 2.10 Exam Pack (University and Review Questions)

☞ **Syllabus Topic : Analysis vs Design**

**Q.** Compare analysis and design.
*(Refer section 2.1)* (4 Marks)

☞ **Syllabus Topic : Class Diagram**

**Q.** What is class?
*(Refer section 2.2)* (2 Marks)

**Q.** What is object?
*(Refer section 2.2)* (2 Marks)

**Q.** Why is a class diagram important in static modeling?
*(Refer section 2.2)* (5 Marks) **(May 2016)**

**Q.** What is class diagram?
*(Refer section 2.2)* (1 Mark)

**Q.** Explain attributes and operation and their representation in class diagram.
*(Refer section 2.2)* (5 Marks)

**Q.** What is association relationship between classes? Take a suitable example and draw the class diagram.
*(Refer section 2.2.1)* (4 Marks)

**Q.** Explain different degrees of multiplicity of association between classes with suitable example.

a. One-to-One Association

b. One-to-Many Association

c. Many-to-many Association

*(Refer section 2.2.1(A))* (6 Marks)

**Q.** Explain aggregation, composition with reference to class diagram. *(Refer section 2.2.2)* (3 Marks) **(May 2017)**

**Q.** Explain generalization with reference to class diagram.
*(Refer section 2.2.3)* (2 Marks) **(May 2017)**

**Q.** Explain the concept of generalization and specialization of class hierarchies using suitable example and class diagram.
*(Refer section 2.2.3)* (5 Marks)

Ex. 2.2.2 (5 Marks) **(Feb. 2016(In sem)**

**Q.** Explain Activity diagram for ATM System. *(Refer section 2.2.5)* (5 Marks) **(Dec. 2016)**

**Q.** Draw a sequence diagram for ATM system. *(Refer section 2.2.5)* (5 Marks) **(May 2016)**

Ex. 2.2.4 (5 Marks) **(Feb. 2016(In sem)**

**Q.** How is class diagram different from an object diagram? *(Refer section 2.2.6)* (3 Marks) **(May 2016)**

☞ **Syllabus Topic : Classes Finding Analysis and Design Classes**

**Q.** Write down the origin or emergence of analysis classes. *(Refer section 2.3.1)* (4 Marks)

**Q.** How to define the analysis class? *(Refer section 2.3.2)* (2 Marks)

**Q.** Write down the anatomy (components) analysis class and design class. *(Refer section 2.3.3)* (2 Marks)

**Q.** Write down the ways of finding the analysis classes. *(Refer section 2.3.4)* (2 Marks)

**Q.** How to find analysis class using noun-verb analysis ? *(Refer section 2.3.4(A))* (4 Marks)

**Q.** Explain finding the analysis class using CRC analysis. *(Refer section 2.3.4(B))* (4 Marks)

☞ **Syllabus Topic : Refining Analysis Relationships**

**Q.** Explain the refinement of analysis relationship. *(Refer section 2.4)* (5 Marks)

☞ **Syllabus Topic : Inheritance and Polymorphism**

**Q.** Define inheritance and polymorphism. *(Refer section 2.5)* (2 Marks)

**Q.** Describe class inheritance. *(Refer section 2.5.1)* (2 Marks)

**Q.** Explain overriding in relation with inheritance. *(Refer section 2.5.1(A))* (4 Marks)

**Q.** Explain abstract operations of classes. *(Refer section 2.5.1(B))* (6 Marks)

**Q.** What is polymorphism? Explain with example. *(Refer section 2.5.2)* (5 Marks)

☞ **Syllabus Topic : Object Diagram**

**Q.** Explain the concept of object diagram. *(Refer section 2.6)* (6 Marks)

☞ **Syllabus Topic : Component Diagram : Interfaces and Components**

**Q.** Describe component based software architecture in brief. *(Refer section 2.7.1)* (6 Marks)

**Q.** Describe component diagram with example. *(Refer section 2.7.2)* (5 Marks)

☞ **Syllabus Topic : Deployment Diagram**

**Q.** Describe deployment diagram with example. *(Refer section 2.8)* (7 Marks)

☞ **Syllabus Topic : Package Diagram**

**Q.** Write use of package. *(Refer section 2.9.1)* (2 Marks)

**Q.** Why it is good practice to group use cases in a use case package? Explain the package diagram with and UML notation. *(Refer section 2.9.2)* (6 Marks)

❑❑❑

UNIT III

CHAPTER 3

# Dynamic Modelling

**Syllabus Topics**

Introduction and Interaction overview diagram, sequence diagram, Timing diagram, Communication diagram, Advanced state machine diagram, Activity diagram.

## Syllabus Topic : Introduction and Interaction Overview Diagram

## 3.1 Overview of Interaction Diagram

**Q.** Explain Interaction diagram in brief. **(6 Marks)**

After the requirement gathering is completed for a proposed software system, the analysis work starts. Use cases are designed and detailed analysis of uses cases are performed. Then will be the time for use case realization in the form of interaction diagram.

- Fundamentally, use case realization is the process of refining the real use case. Interaction diagrams are used for the purpose of use case realization.
- Interaction diagrams are the runtime snapshot of the system. It captures object's interaction scenario of system runtime.
- One of the behavior captures in use case is picked and realized in the form of interaction between analysis class object.
- Interaction diagrams illustrated how object of different classes will interact at run time to implement the specific behavior of specified in use cases.
- So the UML interaction diagram models the collaboration between object to realize one or more system use cases.

☞ **Two types of interaction diagrams**

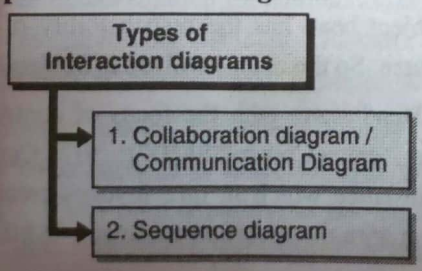


**Fig. C3.1 : Types of interaction diagram**

- So the two view of object interactions are as follows.

➜ **1. Collaboration diagram or Communication Diagram**

This is the structural relationship between the object for analysis of object behavior. It helps in visualization of object collaboration with each other.

➜ **2. Sequence diagram**

The focus of sequence diagram for object interaction is on time ordered interaction in the form of message passing between the objects. This gives better clearer picture of object interaction to understand the system. Sequence diagram will be studied in next section.

## Syllabus Topic : Sequence Diagram

## 3.2 Sequence Diagram

### 3.2.1 Basics of Sequence Diagram

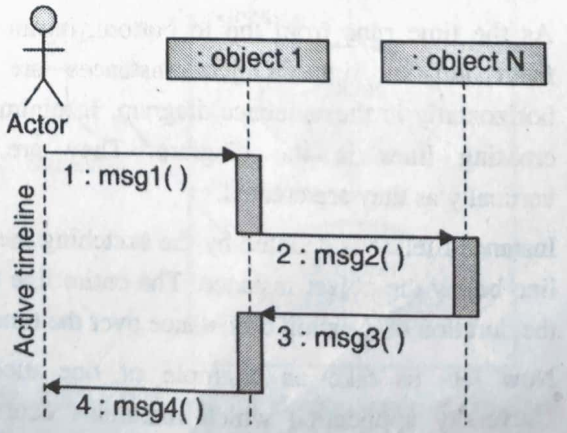


**Fig. 3.2.1 : Generalized overview of sequence diagram**

- Sequence diagrams are used to show to interaction between objects when system becomes operational.

- The sequence of interaction in sequence diagram is decided by messages passed between objects.
- Dynamic view of the system is captured using the sequence diagram. Order of the messaged passed between the objects are shown in the Fig. 3.2.1 starting from top to bottom.
- Numbering of the messages can be shown in sequence diagram (optionally) to show the sequence of messages. Even if not shown, the order of top to bottom shows the order sequencing of message passing.
- Samples of sequence diagrams and state charts are together used to describe the behavioural aspect of system.

### 3.2.2 Sequence Diagram and its Elements : A Use Case Realization

**Q. Explain sequence diagram with its elements. (5 Marks)**

- The section is focusing on sequence with its primary objective of use case realization. As we will progress, we consider one use case and try to formulate the sequence diagram.
- In sequence diagram, object interaction is arranged in time sequence.
- The diagram contains a very unique element called the object lifeline. The interaction diagram shows the objects interaction with other object with respect to the object's lifeline.
- In sequence, the focus of control shifts from one object to another as one object activates other object.
- The primary focus in sequence diagram is the sequencing of events. Time runs top to bottom and events takes place in the time line.
- As the time runs from top to bottom, instances runs from left to right. Object instances are placed horizontally in the sequence diagram. It minimizes the crossing lines in the diagram. They are placed vertically as they are created.
- Instance lifeline is denoted by the sketching the dashed line below the object instance. The entire line denoted the duration of duration of instance over the time.
- Now lets us take an example of one module of University application which maintains courses and subjects for each semester. This application will allow the syllabus to be created for specific courses. Syllabus will be created in the form of units and each unit will have topics. So the application will maintain courses, units, topics, and semesters. System administrator create, update, and delete these elements.

Let us draw a simple use case for new course creation. Course creation will be pre-activity for syllabus creation. The possible use case is shown in Fig. 3.2.2.

| Use Case : Create course |
|---|
| **Actors**<br>• Subject Head |
| **Preconditions**<br>• Subject Head logged in the system<br>• Department is created.<br>• Semester year is created. |
| **Flow of events**<br>• Subject Head selects create course<br>• The system accept new course name<br>• The system updates itself with me new course |
| **Post conditions**<br>• The new course is created in the system |

**Fig. 3.2.2 : Use case for course creation**

- Fig. 3.2.2 shows a use case for course creation in Syllabus Design Module of University Application.
- The possible sequence diagram for the Create Course use case is shown in Fig. 3.2.3.

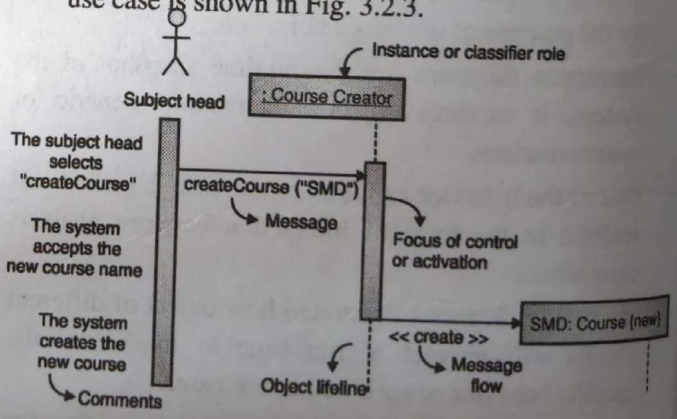


**Fig. 3.2.3 : A sequence diagram for course creation**

- The long rectangle in the sequence diagram signifies that object gets focus of control.
- The subject head has the responsibility of creating a new course. So the actor involved is SubjectHead.
- SubjectHead starts with the focus of control. It sends message createCourse("SMD") to :CourseCreator object instance which in turn invokes createCourse( ) operation with the specified parameter "SMD".

- The actual course creation with the name "SMD" is specified by passing the stereotyped message <<create>>. This is equivalent to calling the constructor (in many programming languages like java, C++, etc.) for creating course object with parameter of course name i.e. SMD.
- Additional element that can be explored for sequence diagram is the **iteration.** Iteration takes place when set of messages are getting repeated. Let us take an example of adding units in the course.

  Once the course has been created multiple units can be added in the courses. The iteration of repeatedly adding the units in the course is shown in Fig. 3.2.4.

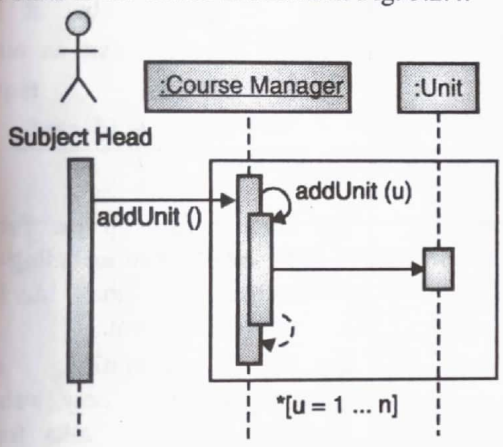


**Fig. 3.2.4 : Sequence diagram with iteration**

- The large rectangle box shows the iteration of unit creation and addition by the :CourseManager. This activity is shown by curved arrows in the Fig. 3.2.4. Also the maximum of n units can be added is specified in the iteration.

### 3.2.3 Case Study for Sequence Diagram

- Now, we will try to discuss each few more aspects of sequence diagram in relation with a practical scenario. We are taking a case study of university application.
- We will be considering a virtual University for which we want to prepare a web based application with several functionalities. It will maintain student and teacher activities and accessible remotely.
- It will complete solution for automation of all academic activities involving student ID-card, attendance, examination, admission, leaves, notices, feedback etc.
- The obvious choice for design methodology will be object orientation as it has clear real world problem domain. It has been experienced and realised over the years that real world automation problems are best addressed by object oriented methodology.
- Understanding the actual process of University activities, identifying the objects in problem domain are few primary steps that we have to perform in object oriented analysis.
- For simplicity, we will be considering following required services needed by the University Application. We have assigned these activities to dedicated **subsystems** :

1. Student_Id_Card
2. Faculty Id-Card
3. Lecture Updates
4. Attendance
5. Examination
6. Leave
7. Notices
8. Feedback
9. Application Administration

- As we will discuss different notation and terms of object oriented designing and analysis, we will keep on refining our virtual University Application requirements.
- So for now, we are not freezing all the requirements of University Application. As we will progress, will specify the requirements, use case scenario and will explore it.
- Now lets us draw the use case for specific system module called the Student ID card subsystem for student record creation and verification. Fig. 3.2.5 shows the use case for student record creation.

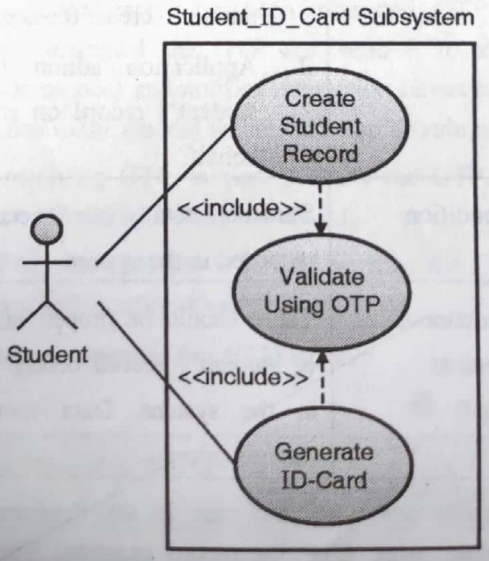


**Fig. 3.2.5 : Use case for student record creation**

- The formal description of the use case is provided shown in Table 3.2.1. It specifies actors, summary, precondition and post-condition and sequences.

**Table 3.2.1**

| Use Case Name | Create Student Record |
|---|---|
| Actors | Student, Application Administrator |
| Summary | A student can create the individual record by clicking on New User/Register link on home page and filling up the registration form by entering a guest username password provided by university. |
| Precondition | Student should have a temporary guest username and passwords provided at the time of admission, which student can use to login and create the record. |
| Description Primary Sequence | 1. User opens the home page and clicks on New User / Register link.<br>2. Then provides a temporary username and password and after that can fill the form, upload the photograph and submit. |
| Description of Alternative Sequence | 1. Student does not have guest login and password; He/She directly approach application admin to create the account.<br>2. Application admin fill the student's record on students' behalf. |
| Post Condition | Students details are successfully recorded in the system. |
| Non Functional Requirement | There should be proper validation of student's record before storing in the system. Data should be securely stored. |

- There is one more use case in the diagram named validate using OTP for record creation. The formal description of the use case is provided shown in Table 3.2.2.
- It specifies actors, summary, precondition and post-condition and sequences.

**Table 3.2.2**

| Use Case Name | Validate using OTP |
|---|---|
| Summary | Validation of student mobile number and email id before submitting the record for approval |
| Actor | Student |
| Precondition | **Case 1 :** Students record is not submitted for approval and mobile number and email id need to be verified<br>**Case 2 :** Student record is already approved but it is not verified that an authentic student is requesting duplicate Id- card |
| Primary Sequence | **Case 1**<br>1. Student fills up the form for personal record including mobile number and email id. Student submits the form.<br>2. After submitting, student navigates to a new web page. New web page asks for OTP received on mobile number.<br>3. System generates the OTP and sends it to student's mobile phone.<br>4. Student receives the OTP on his/her mobile phone and enters it in requesting page.<br>5. Student gets the response message depending on correct or incorrect OTP.<br>6. In case of correct OTP, student gets a message to click on the link sent to the email id.<br>7. Student clicks the link received on email and link navigates to a page saying your record is submitted for approval.<br>**Case 2**<br>1. Student requests for a duplicate id by clicking on generate duplicate Id-card.<br>2. After clicking, student navigates to a new web page. New web page asks for OTP received on mobile number. |

| Use Case Name | Validate using OTP |
|---|---|
| | 3. System generates the OTP and sends it to student's mobile phone.<br>4. Student receives the OTP on his/her mobile phone and enters it in requesting page.<br>5. Student gets the response message depending on correct or incorrect OTP.<br>6. In case of correct OTP, student gets a message to click on the link sent to the email id.<br>7. Student clicks the link received on email and link navigates student to a new page where student can print the duplicate ID card. |
| Alternate Sequence | 1. Student is navigated to the page where he/she has to submit the OTP.<br>2. Student has not received the OTP and after timeout period student clicks on link Problem in Receiving OTP.<br>3. After clinking the link student verifies the email id as specified in primary sequence.<br>4. After verifying email id , student gets the message that, form has been submitted for approval and but Mobile Number is not verified.<br>5. System marks a flag in the student's record that Student's phone number is not verified.<br>6. Admin follows the manual process for verifying student's mobile number. |
| Post Condition | Student's record gets submitted successfully for admins approval with auto verification of valid mobile number and email id.<br>In an alternate case, record is submitted but marked as unverified mobile number. |

- Fig. 3.2.6 shows the sequence diagram for Student Record Submission and OTP verification.

☞ **Explanation of sequence diagram**

- Student login to interact with system by invoking login operation of the object representing system's interaction interface.
- The interaction object is marked <<interaction>>. This object represents GUI object that offers the user a form to submit the record and capable to communicate with backend system and OTP service.

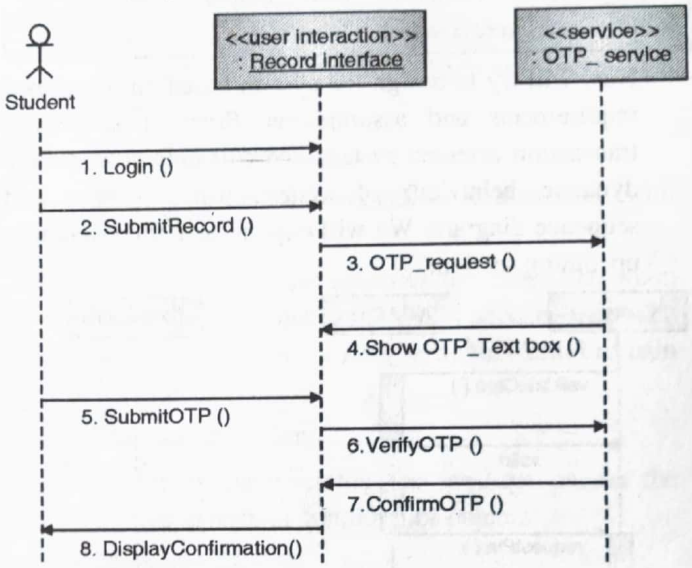


**Fig. 3.2.6 : Sequence diagram for student record creation**

- Once record submitted, OTP service object get the message that record has been submitted. OTP service object generated the OTP and send it to student's mobile number and notifies interaction object to shoe a text box to the student where OTP can be submitted.
- On receiving OTP, student submits the OTP and on screen offered by interaction object.
- OTP service gets the OTP and verifies the OTP and replies with confirmation.

☞ **Sequence diagram for ATM system**

➔ **(May 2016)**

**Q.** Draw sequence diagram for ATM system.
**SPPU- May 2016, 5 Marks**

- Another example of system with transaction oriented operation is ATM system that connects user to the bank for performing different transactional operation.
- The requirement is briefly explain here.

**ATM System : Requirements**

ATM system is a mini banking system for customers of same or different bank. It uses an automated teller machine that validates customer's ATM card and connects to bank server to perform different transactional operations. It has local repository of cash that can be withdrawn on successful request acceptance. User interacts with ATM machine using an interactive control panel screen. User can validate card and pin, withdraw cash, check mini statement etc. ATM client system serves as an interface for different banking operation. ATM client connects with banking server for transactional operation.

- Now will try to design the system based on mentioned requirements and assumptions. Since it is heavily transaction oriented system, we will only consider the dynamic behaviour of system using activity and sequence diagram. We will explore activity diagram in upcoming section.

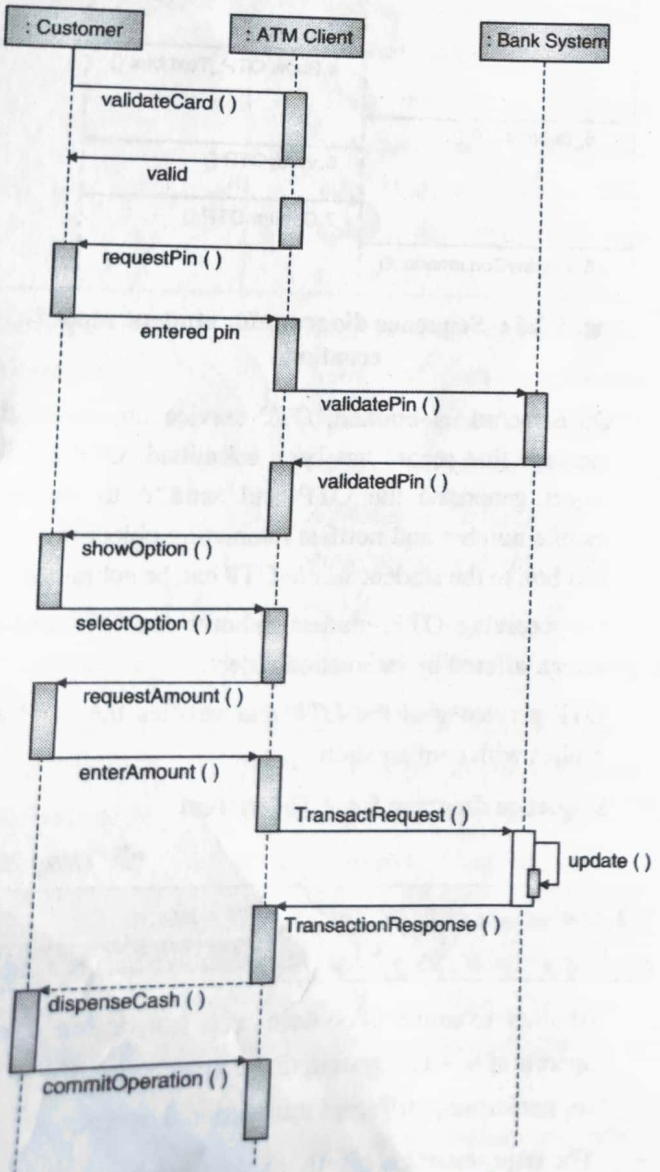


**Fig. 3.2.7 : Sequence diagram for ATM system**

- Let us try a simplified sequence diagram online shopping system. For assignments, students can extend this simplified sequence diagram to add more components and activities like iteration, constraints, branching, etc.

**Ex. 3.2.1** SPPU - Feb. 2016(In sem), 5 Marks

Draw a sequence diagram for Online shopping system.

**Soln. :**

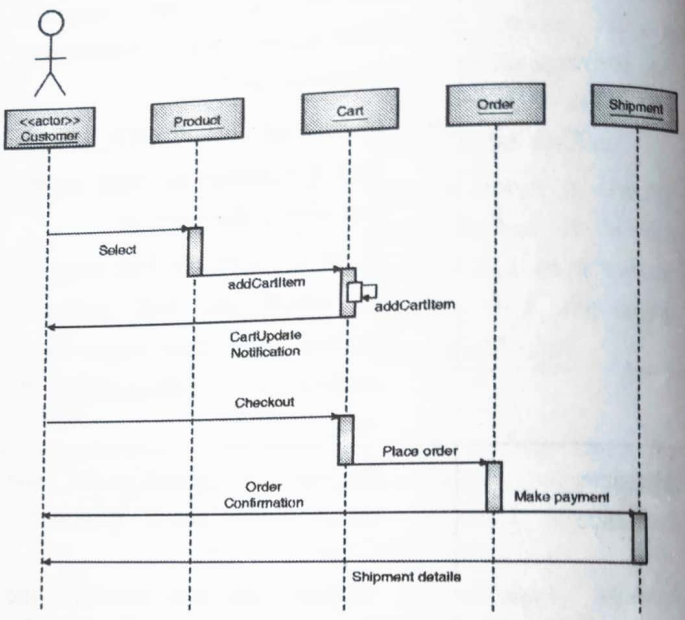


**Fig. P.3.2.1 : Sequence diagram for Online Shopping System**

**Ex. 3.2.2**

Draw the sequence diagram for e-learning system with following requirements.

Recently a well know publisher has launched their e-learning service for students. Users of the system will be system administrator, course creator and students. Administrator will manage the system and start the initial system setup and configuration. Creator will create the video lectures for the courses assigned by administrator. Students can register, login download and subscribe the video lectures by paying online.

Following is the work flow of the system :

1. Administrator creates and configures all the courses (subjects)
2. Administrator assigns each course to one or more creators.
3. Creators can record video lecture and upload in the system
4. Creator can only upload for assigned courses.
5. Students can login, register and subscribe the course by making online payment.

Student can download(stream) all the videos lectures for the course he/she has subscribed.

**Soln. :**

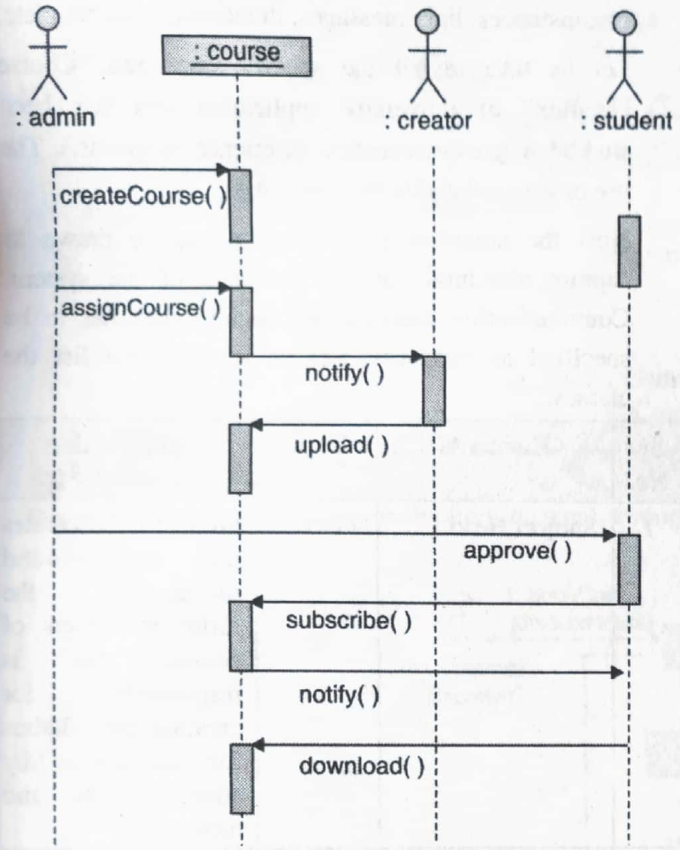


**Fig. P. 3.2.2**

**Syllabus Topic: Timing Diagram**

## 3.3 Timing Diagram

**Q. Explain timing diagram in brief. (4 Marks)**

- UML's Timing diagram in one way are extension of sequence diagrams where the focus of design is on time constraints.
- Few of the basic concepts related to timing diagram are shown in Fig. C3.2.

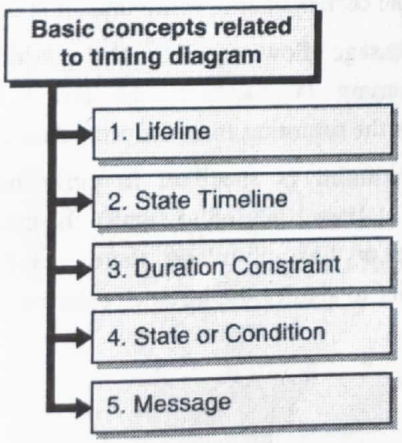


**Fig. C3.2 : Basic concepts related to timing diagram**

➜ **1. Lifeline**

- Lifeline creates a rectangular space within the content area of a frame.
- Each lifeline is a named element. It represents participation of the element in the interaction.
- The alignment of lifeline is horizontal and it is read left to right. Same frame can have multiple lifelines.

➜ **2. State Timeline**

- A state or condition timeline specifies the set of valid states and time.
- The states are placed on the left margin of the lifeline from top to bottom. They are stacked one on another.

➜ **3. Duration Constraint**

- Each state or value transition is associated with well defied events.
- The time constraint specifies the exact that when should be the event occur and a duration constraint specifies how long a state or value should remain in effect.

➜ **4. State or condition**

- Any even on receiving the message causes the change in state or condition of object.

➜ **5. Message**

- The change in condition happens on well-defined events in the system.
- The event occurs on delivery of message in the form of operations of object.

Let us draw one possible timing diagram for the sequence diagram we have studied in previous section. Consider the sequence diagram Fig. 3.3.1.

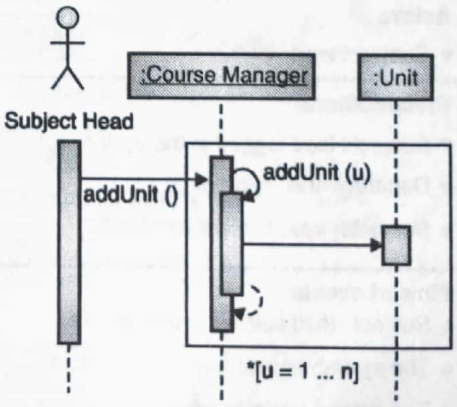


**Fig. 3.3.1: Sequence diagram for adding repeatedly the units in the course**

- The corresponding timing diagram for the specified sequence diagram of unit creation is shown below in Fig. 3.3.2.

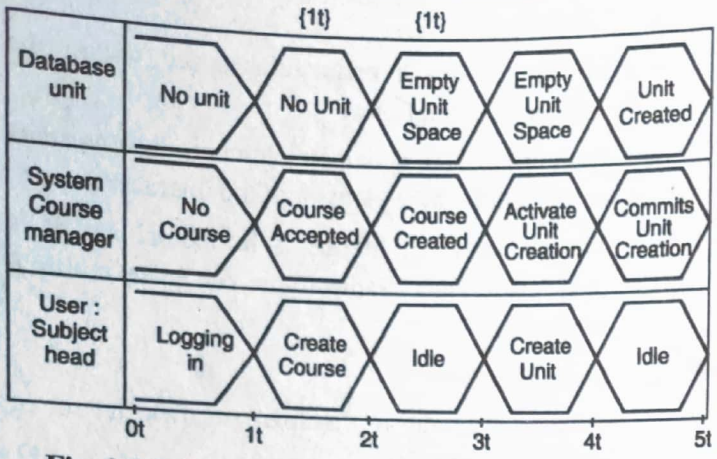


**Fig. 3.3.2: Timing Diagram for unit creation for the specified course**

## Syllabus Topic : Communication Diagram

## 3.4 Communication Diagram

**Q.** Explain communication diagram with example. **(6 Marks)**

- Communication diagram (or collaboration diagram) is the kind of interaction diagram which captures the structural run time behavior of the system. Communication diagram focuses on communication and messages between the object instead of focusing on timing.
- Messages however can be ordered (numbered) to signify the ordering in which they are passed and events occur.
- Communication diagram focuses on structural aspect of the object interaction.

| Use Case : Create course |
|---|
| **Actors**<br>• Subject Head |
| **Preconditions**<br>• Subject Head logged in the system<br>• Department is created.<br>• Semester year is created. |
| **Flow of events**<br>• Subject Head selects create course<br>• The system accept new course name<br>• The system updates itself with me new course |
| **Post conditions**<br>• The new course is created in the system |

**Fig. 3.4.1 : The use case for course creation**

- The various elements of the communication diagram are: instances, link, messages, iterations, branching. etc.
- Let us take revisit the specific use case "Course Creation" of university application that has been studied in previous section (sequence diagrams.). The use case is restated in the Fig. 3.4.1.
- Now the communication diagram can be drawn to capture structural runtime behavior of the system. Communication diagram will require classifier to be specified as instances. Let us identify and list the classes.

| Sr. No. | Elements | Type | Specification (Semantics) |
|---|---|---|---|
| 1 | Subject Head | Actor | Subject Head creates the course and manages the different aspects of course. He is responsible for creating the syllabus for the course by adding units and topics. |
| 2 | Course | Class | Contains all details of course including all its attributes |
| 3 | CourseCreator or CourseManager | Class | This is the dedicated functional class encapsulating all the operation course creation and management. |

- For all the above specification including use cases, the communication diagram can be drawn. It will be the realization of the specified use case. The Fig. 3.4.2 shows the corresponding communication diagram.
- The massage flow signifies the procedure call in programming language. If the arrow is dotted it signifies the returning from the procedure call.
- The constraint is specified in curly brackets. The constraint {new} is used to signify the creation of new instance or link. Similarly there can be {destroy} constraint to signify the instance deletion or interaction deletion.

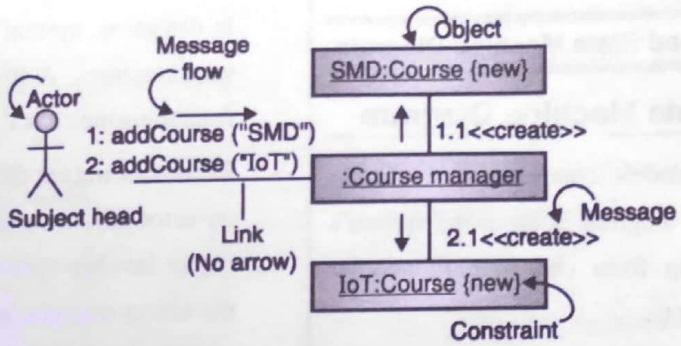


**Fig. 3.4.2 : Communication Diagram for Course Creation**

**Ex. 3.4.1**

Draw the communication diagram for booking a cab through interactive voice response system of cab service provider.

**Soln. :**

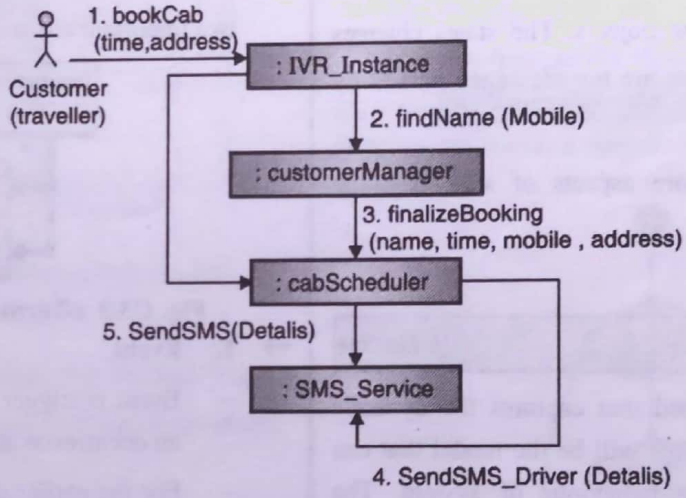


**Fig. P.3.4.1(a) : Generalized Communication diagram for e-cab**

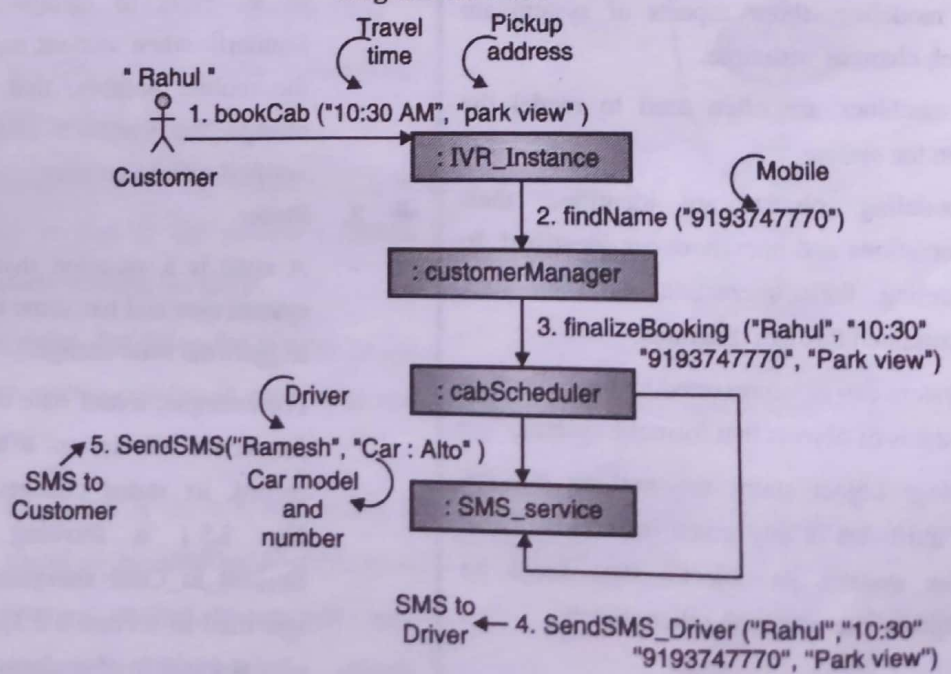


**Fig. P.3.4.1(b) : Instance specific Communication diagram for e-cab**

**Syllabus Topic : Advanced State Machine Diagram**

## 3.5 Advanced State Machine Diagram

- Another construct that models dynamic behavior of the system is state machine diagram. It focus the system's machine view switching from one state to another based on predefined events.
- Lifecycle of the single reactive object is modelled using the state machine diagram. Reactive object provides context to the state chart state machine.
- There can be one state machine for each class in the system. This machine models all the transitions between states of all the objects. The states changes because of events. Events are the messaged passed by other objects.
- Let us discuss few more aspects of state machine diagram.

### 3.5.1 State Chart

**Q. Explain state chart in detail. (6 Marks)**

- A model can be designed that captures the dynamic aspects of the system. This will be the model that can illustrate the rum time behaviour of system. The process is called dynamic modelling.
- In dynamic modeling, those aspects of system are captured which changes with time.
- Finite state machines are often used to model the control flow in the system.
- In static modeling, objects are identified; their attributes associations and operations are identified. In dynamic modeling, their interaction and their state changes are explained through diagrams.
- State of the system can be represented by the collection of individual states of objects that form the system.
- In programming, object states are nothing but the values of the attributes at any given instance of time. For a dynamic system, its object's state keeps on changing as objects interacts with other objects.
- In designing, system state can be represented by finite state machine. A finite state is a virtual or conceptual machine which has finite number of states.
- Event can trigger the state change. For example when an actor invokes some operation in an object or one object invokes operation of other object then state of the object changes, and this behaviour can be captured in design by finite state machines.
- Change is object states can be depicted as,state transition from one to another.
- In object oriented designing, the objects' finite state machine is often depicted by **state charts**. There are two important terms related with state charts.

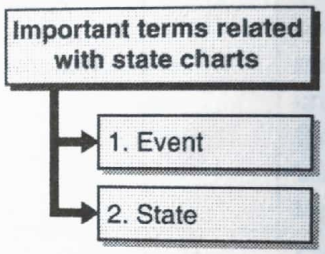


**Fig. C3.3 : Terms related with state charts**

➜ **1. Event**

- Event is trigger that causes the state change. It is an occurrence at any particular time.
- For the earlier discussed case study of University Application, when submits the record, that serves as an event to change the state of system. Similarly when student enters the OTP to verify the mobile number, that is an event that can change the system's state from unverified to verified mobile number.

➜ **2. State**

- A state is a situation that is recognized by the system user and has some time duration. An event triggers the state change.
- For example, initial state of the system is waiting for record submission. When student submits the record, its stated changes to waiting for OTP. Fig. 3.5.1 is showing the state chart for Student_Id_Card subsystem (The case study is specified in section 3.2.3). Here we are capturing partial scenario of students record submission.

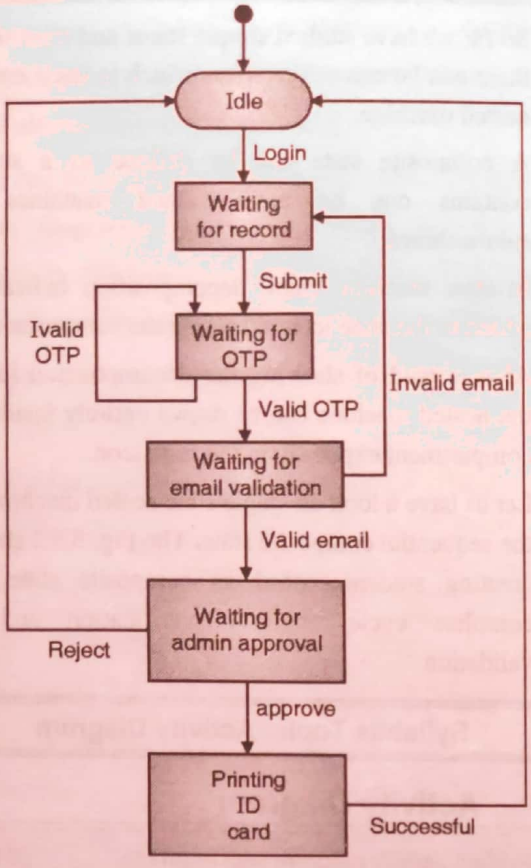


**Fig. 3.5.1: State chart for student record creation and Id card generation**

- Fig. 3.5.1 shows the state chart for Student Record Creation that also includes Student Id-card generation.

**States and events**

- System remains in idle state until some new student logins to create the record by filling up the web form.
- Login is an event that changes the system's state from **"idle"** to **"waiting for record submission".**
- After submission of record, system generated OTP (one time password) that is sent to the student's mobile phone and system starts **waiting for OTP**.
- An invalid OTP is event that takes the system to the same state. It means system remains in same state and still waits for valid OTP.
- A valid OTP takes the system to the new state where it checks for correct email id by sending a verification email. In case, id is not get verified, system will come back to earlier state of ***"Waiting for record submission"***.
- In case of correct email id, system will switch to another state where it will wait for admin's action to approve.
- If admin finds the records inappropriate, he or she will reject the approval request system will go back in idle state.
- On successful approval, system will generate the Id-card of approved student and finally will go back to idle state.
- Same control flow and sequencing will get repeated each time a new student will create the record.
- Let us take one more example of simplified state machine diagram for vending coffee machine

**Ex. 3.5.1** **SPPU - Feb. 2016(In sem), 5 Marks**

Draw a state machine diagram for coffee vending machine.

**Soln.:**

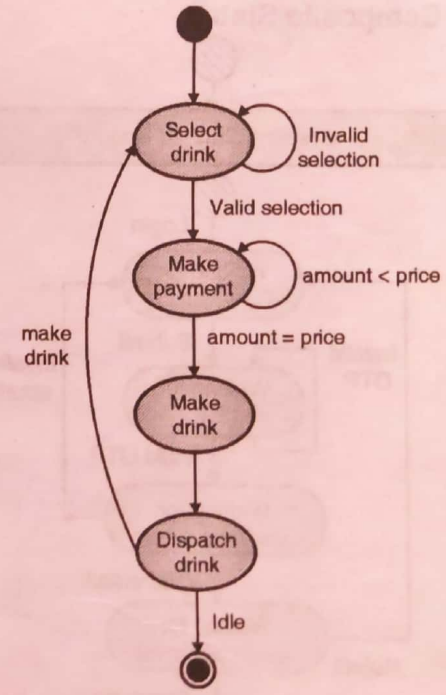


**Fig. P.3.5.1**

**Ex. 3.5.2**

Draw the state machine diagram for IoT based home security system including the movement sensor at door and camera as detection device.

**Soln.:**

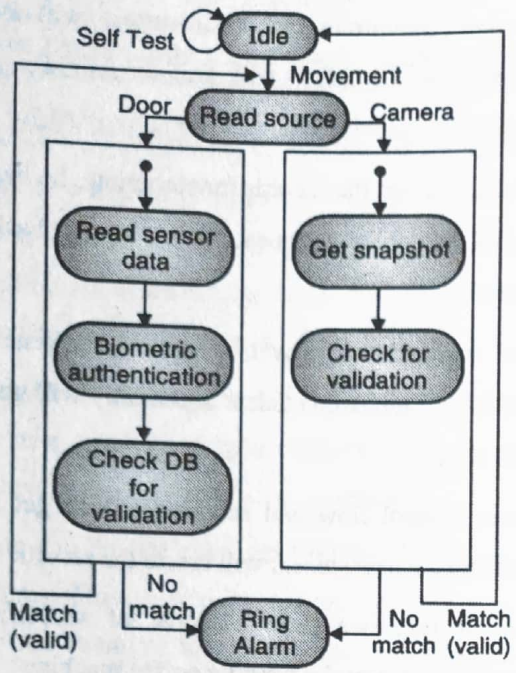


**Fig. P. 3.5.2 : State machine diagram for Home security system**

### 3.5.2 Advanced State Machine Diagrams : Composite States

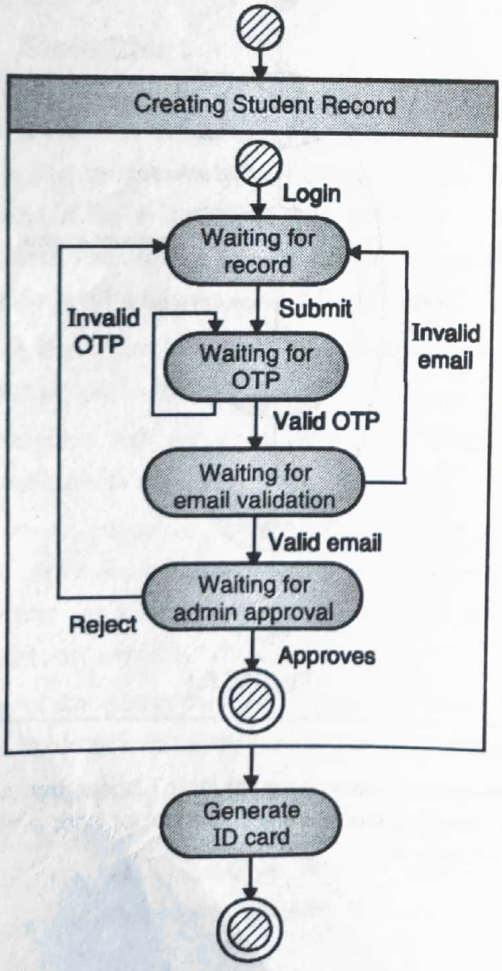


**Fig. 3.5.2 : State Machine Diagram with composite state**

- So far we have studied simple states and transition. But there can be composite states which in itself contains a nested machine.
- A composite state can be defined as a state that contains one or more nested machines called submachines.
- In state machine chart, decomposition indicators are added to the state icon to specify the composite states.
- Also instead of showing the decomposition indicator, the nested machine can be drawn entirely inside a new compartment explicitly in the state icon.
- Let us have a look on single state nested machine called the sequential composite state. The Fig. 3.5.2 shows the creating student record as composite state having complete cycle of OTP verification and email validation.

## Syllabus Topic: Activity Diagram

## 3.6 Activity Diagram

**Q. Explain activity diagram with example. (5 Marks)**

- Activity diagrams are used to represent flow of different activities in the system or sub system.
- Activity diagrams are standard UML diagrams and have well defined notations.
- It shows the flow of control and sequencing among different possible activities.
- Like use case, activity diagram also captures the user's perspective of system or sub system
- Activity diagrams do not explore system design or system's processing logic, instead focuses on flow of activities that a user of the system can experience.
- Activity diagrams are mostly used for work flow modeling in Web Services and SOA (Service-Oriented Architecture) applications.
- In relation with our previous discussion of Use cases, we can think activity diagram as organizing different uses case in a chronological order, depicting the flow of control in system.
- While arranging the uses cases chronologically, both the step sequences of use cases (primary and

alternative) are taken into consideration. A decision making point is used to denote the place where primary and alternative path get separated.

- We can think activity diagrams as refinement over Use case diagrams, where we capture scenario more precisely. We exactly describe the point, where primary and alternative sequence of activities gets separated.
- What is the condition that forces the control to flow in primary or alternative direction.
- Use cases become the nodes in activity diagrams and are called the activity nodes.
- Others are decision nodes which are used to denote the point where decisions have to be made.
- For example, take the Student_Id_Card subsystem and try to generate the activity diagram for the activities of Creating Student Record, Approving Student Record and Generating ID Card. A formal Use Case description table has been provided in section 3.2.3.

☞ **Primary sequence of activities**

- The activity diagram is shown in Fig. 3.6.1.
- For the activity diagram shown in Fig. 3.6.1, the **primary** sequence of activities can be described as :
  - Student fills up the online form for ID card including email Id and mobile number.
  - Student submits and receives the One Time Password (OTP) on the mobile number.
  - Student enters the correct OTP and then validates the email Id.
  - After successful validation, application admin approves the record.
  - Admin can generate the Id-card for student.

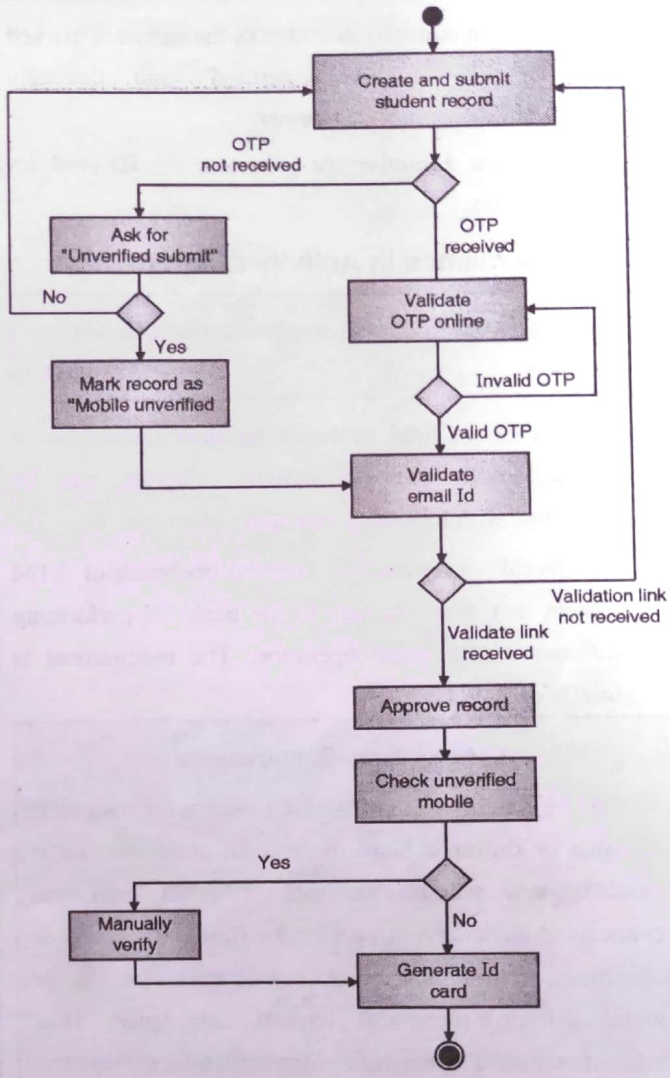


**Fig. 3.6.1 : Activity Diagram**

☞ **Alternative sequence of activities**

- Similarly, the **alternative** sequence of activities can be described as follows:
  - Student fills up the online form for ID card including email Id and mobile number.
  - Student submits record but does not receives the One Time Password (OTP) on the mobile number.
  - After time out period, system asks student whether he/she wants to continue by leaving mobile number unverified or specify the correct mobile number again be resubmitting the form.
  - In case student continues, system marks student record as "mobile number unverified" and navigates to the email verification page.
  - After successful email verification, system administrator approves the record.

- System administrator checks the record is marked "Mobile number unverified" and manually verifies the mobile number.
- System administrator generates the ID card for student.

### 3.6.1 Swimlanes in Activity Diagram

**Q. Explain the Swimlanes in activity diagram with suitable example. (6 Marks)**

- Swimlanes are used to depict the concurrent flow of activity. Swimlanes in activity diagram can be explained with following example.
- Let us take a transaction oriented operation of ATM system that connects user to the bank for performing different transactional operation. The requirement is briefly explain here.

**ATM System : Requirements**

ATM system is a mini banking system for customers of same or different bank. It uses an automated teller machine that validates customer's ATM card and connects to bank server to perform different transactional operations. It has local repository of cash that can be withdrawn on successful request acceptance. User interacts with ATM machine using an interactive control panel screen. User can validate card and pin, withdraw cash, check mini statement etc. ATM client system serves as an interface for different banking operation. ATM client connects with banking server for transactional operation.

- Now will try to design the system based on mentioned requirements and assumptions. Since it is heavily transaction oriented system, we will only consider the dynamic behaviour of system using activity diagram.
- The activity diagram that illustrates the normal operation is depicted in Fig. 3.6.2. Three Swimlanes are customer, ATM system and bank system.
- The activity diagram is divided between the activities flows or Swimlanes of customer, local ATM subsystem and bank system.

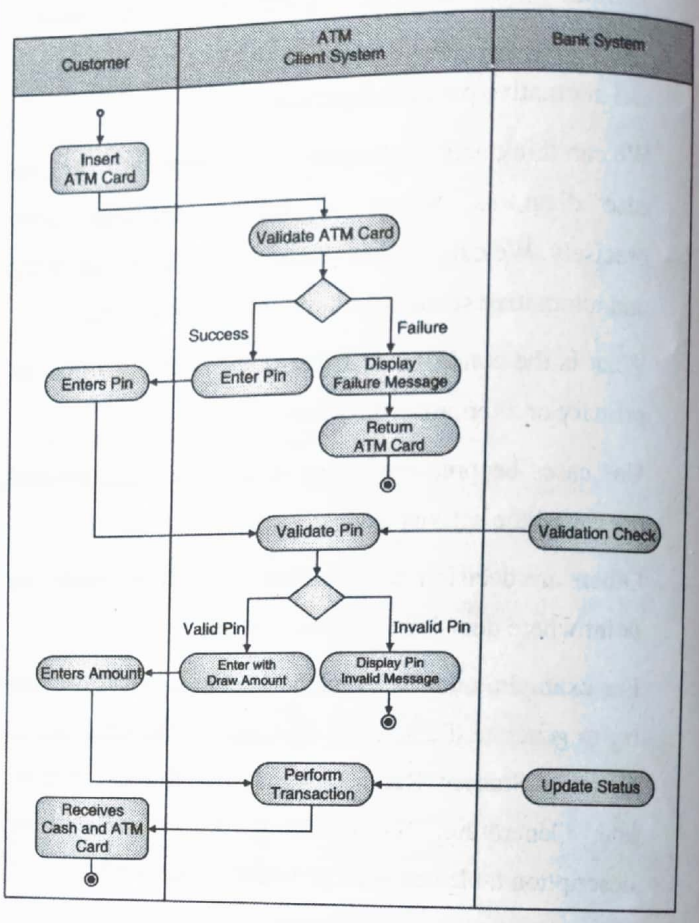


**Fig. 3.6.2 : Activity Diagram for ATM system**

### 3.6.2 Forks and Joins in Activity Diagram

- Concurrent flow in activity diagram can be achieved by splitting activity flow into two paths using forks and can be synchronized using join.
- A fork splits the activity into two or more concurrent flows. It has exactly one incoming transition. It has two or more outgoing transition
- A join synchronizes two or more concurrent flows. Join has two or more incoming transitions. It has only one outgoing transition.
- Both are quite useful in designing the dynamic behavior of concurrent systems.
- Let us draw the simple activity diagram with fork and join to illustrate the concept. The Fig. 3.6.3 shows such activity diagram syllabus creation or University application system.

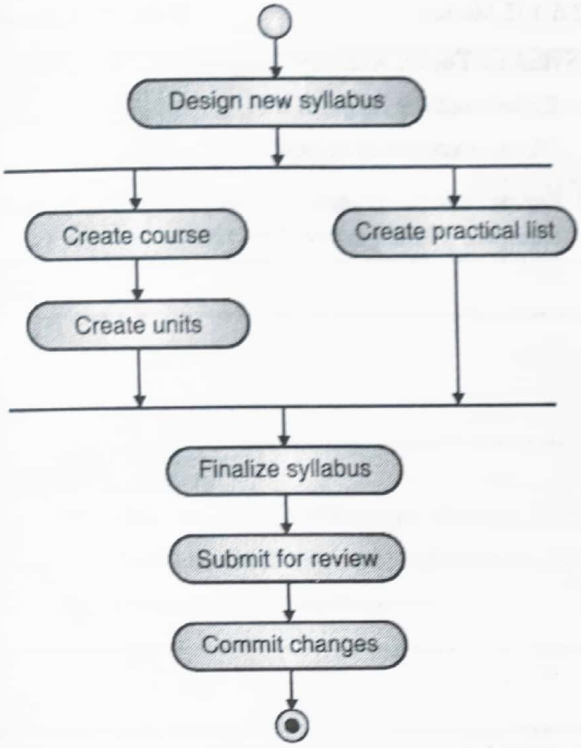


**Fig. 3.6.3 : Activity Diagram of Syllabus Creation**

## Summary

- **Object orientation** is way of thinking a problem with real world perspective. The concept of the system in mind is mapped in the objects, their properties behaviour and interaction.
- **Object orientation** avoids the problems and pitfalls associated with conventional designing methodology.
- The concepts that we mostly address in **object orientation** approach is classes, objects, inheritance, generalization, specialization, polymorphism and abstraction.
- **Objects** are the central concept in object orientation. It represents the abstraction of any real object in problem domain. It reflects the capability of the system. It reflects what information system will keep and also it reflects what methods it offers to interact with the system.
- Information hiding or **abstraction** is another central approach of object oriented designing. If a module is well contained with its own information, it will be less dependent on other modules and hence changes can be easily done without affecting the other modules.
- **Object orientation analysis** is the first method applied in object oriented methodology of software development. This is the analysis phase of software development. The mapping of real world to programming world is performed here. Objects are identified along with their properties and behaviour. The phase is often called **object modeling**.
- **UML** (Unified modeling Language) is standard modeling language and notation for object oriented analysis and design.
- **OMG** (Object management group) has adopted UML as standard notation for designing in 1997. The most significant contributors in forming the notations and specifications of UML are Grady Booch, Ivar Jacobson and James Rumbaugh. The design and notation of UML was developed by them at Rational Software.
- **Use cases** capture the functionality of the system from user's perspective. It is represented as the sequence of interaction between **actor** and the **system.**
- Use case diagram was proposed by Jacobson in the book subtitled *"A use case driven approach"* 1992.
- Use cases are described in texts as well in diagrams. Text form of use cases are called ***use case descriptions.***
- **Activity diagrams** are used to represent flow of different activities in the system or sub system.
- **Activity diagrams** are standard UML diagrams and have well defined notations.
- **Activity diagrams** shows the flow of control and sequencing among different possible activities.
- **Activity diagrams** do not explore system design or system's processing logic, instead focuses on flow of activities that a user of the system can experience.

## 3.7 Exam Pack (University and Review Questions)

☞ **Syllabus Topic : Introduction and Interaction Overview Diagram**

Q. Explain Interaction diagram in brief.
*(Refer section 3.1)* (6 Marks)

☞ **Syllabus Topic : Sequence Diagram**

Q. Explain sequence diagram with its elements.
*(Refer section 3.2.2)* (5 Marks)

Q. Draw sequence diagram for ATM system.
*(Refer section 3.2.3)* (5 Marks) **(May 2016)**

Ex. 3.2.1 (5 Marks) **(Feb. 2016(In sem))**

☞ **Syllabus Topic : Timing Diagram**

Q. Explain timing diagram in brief.
*(Refer section 3.3)* (4 Marks)

☞ **Syllabus Topic : Communication Diagram**

Q. Explain communication diagram with example. *(Refer section 3.4)* (6 Marks)

☞ **Syllabus Topic : Advanced State Machine Diagram**

Q. Explain state chart in detail. *(Refer section 3.5.1)* (6 Marks)

Ex. 3.5.1 (5 Marks) **(Feb. 2016(In sem))**

☞ **Syllabus Topic: Activity Diagram**

Q. Explain activity diagram with example. *(Refer section 3.6)* (5 Marks)

Q. Explain the Swimlanes in activity diagram with suitable example. *(Refer section 3.6.1)* (6 Marks)

❑❑❑

UNIT IV

CHAPTER 4

# Architecture Design

**Syllabus Topics**

Introduction to Architectural design, overview of software architecture, Object oriented software architecture, Client server Architecture, Service oriented Architecture, Component based Architecture, Real time software Architecture.

**Syllabus Topic : Introduction to Architectural Design**

## 4.1 Introduction to Architectural Design

**Q. What is architectural design of a software system? (4 Marks)**

- Software Architecture or the architectural design of software is an overall architecture of the software system to be designed. This represents the entire system in the form of sub systems and their interconnection.
- It may represent the internal details of the individual sub-system and their communication interfaces with other sub systems. The design of systems starts by analyzing requirement and thinking of a high level system structure based on functionality.
- A designer or system architect first starts thinking about the modular system that can perform the required task. With its inception in architects mind, systems architecture starts taking its shape and form with help the of well defined and standardized designing constructs.
- Such designing constructs are well defined by UML and are universally accepted. Software architecture is mainly used to refer the high level architecture of the system and serves as the foundation for all subsequent design and implementation activities.
- In 2003, a published work of Kazman, Bass and Clements defines the software architecture as follows:
- "The software architecture of program or computing system is the structure or the structure of the system which comprise software elements, the externally visible properties of those elements and the relationship among them."
- It should be noted however that architecture at the beginning is prone to change and can reform itself with subsequent design activities. In many scenarios of software development it is desired that software architecture should be frozen first and can be referred for all subsequent design activities.
- Freezing the architecture at early stage however leaves less scope for design independence and associates a risk of failure. But change in architecture at later stage of development may incur a significant cost in terms of time, effort and man power but to gain the competitive advantage, no organizations rule out such reformation in software architecture in modern era of software development.
- Despite of all the arguments, in reality if an organization always keeps the habit of an early design of software architecture and rarely needs to change the architecture for subsequent stage of software design and implementation, it shows that the organization has worked well for understanding the requirement, they are disciplined in design process and have understood the importance of software architecture.

- So the conclusion says that if you are an expert and experienced software architect and have understood the problem domain very well, you will have the confidence to freeze the architectural design at early stage. But if you are experimenting or dealing with entirely a new system, and your requirements are prone to change, it better to leave the scope of change in architectural design.

## Syllabus Topic : Overview of Software Architecture

### 4.1.1 Overview of Software Architectural Design

**Q. Explain overview of architectural design. (2 Marks)**

- The architectural design is often termed as software architectural design or called just as the software architecture.
- Software architecture depicts a modular design of the system in the form of components and their interconnection. Software architecture also refers the detailed internal design of each module and their interfaces.
- An architecture design for software system that focuses on components and their interconnection is called "Programming-in-the-large".
- An architectural design that focuses on internal design of each component is called "Programming-in-the-small".
- Architectural design covers different aspects of the system at different level of details. One over all architecture depicts entire system as a whole. Another can decomposes the system into subsystems and shows the interaction.
- Also there could be more architecture to show the modular decomposition of each sub-system and so on.
- The design principles and specifications are applied throughout the system architecture design process.
- At each level there are well defined process and notations to implement the design that is universally accepted.
- These processes, design methods and notations can be combined to implement the creativity and innovation in Software architectural design and results in a successful product.
- One of the most important factors of software system is its quality attributes like performance, maintainability and security.
- These quality attributes falls under the category of non-functional requirements.
- A good architectural design always tries to convince the viewer that system will satisfy these requirements.

### 4.1.2 Importance of Architectural Design

**Q. Explain the importance of architectural design. (4 Marks)**

- Designing the architecture of software system is the beginning of software implementation. This is the crucial stage and whatever design decision made at this stage affects the entire implementation of system.
- Architectural design is of great importance as it represents the entire system in brief. One can refer it and will get the high level idea of what system need to be developed and how the system will work.
- A usual approach is to capture the static design first which mainly highlights the structure of system using the class diagram. Here system is divided into subsystems or modules which are depicted by class diagram. Class diagrams mainly describe *what* system needs to be developed.
- The dynamic behaviour of system can also be captured in architectural design. It describes in brief that how the system should work.
- However, the degree of description of *how* is a major design decision. It means it is not always possible at beginning of architectural design that we can decide how system will work.
- Architectural design provides multiple views of software architecture capturing the system characteristics from different perspectives.
- Different stake holders may wish to view the system from different perspectives. Different views will be covered in next section.
- It offers structures design to capture the static characteristics of the system which are depicted by class diagram.
- It offers dynamic design to capture runtime behaviour of the system which are depicted by communication, sequence or collaboration diagrams.
- It offers deployment design to specify which modules will get placed at which location.

- In modern networked and distributed environment, a deployment design exactly describes which node hosts which subsystem without bothering about internal design of system. Deployment diagrams are used for the purpose.
- An architectural design provides the foundation for all subsequent design activities.
- A small flaw in architectural design may lead the entire design and implementation activity in entirely different direction.
- Architectural design provides a link between system requirement analysis and system implementation.
- A good design will translate or convert all the specified requirements into a successful system.
- Architectural design with the help of predefined, well practised design patterns can make the system implementation very disciplined, easier and smooth. (Design patterns will get discussed in next chapter).

### 4.1.3 Component Based Software Architecture

**Q. Describe component based software architecture in brief. (5 Marks)**

- One of the major concepts associated with software architecture is component based software architecture. This is the structural perspective of software design.
- In component based software architectural design, the primary focus is in dividing the system in components. Each component is well defined self contained block of system.
- A component encapsulates all the complex information about itself inside it. So, a component does not describe how the system or the component works internally but it specifies what responsibility it has.
- A component can be a simple object like Student object or it can be a composite object like Department object containing other simple objects.
- A component also specifies the interface through which it communicates with other components.
- One component does not need to know all the information about the other to communicate. It just need to know what methods are offered by the other component's interface to communicate.
- For example **Student** object may offer an interface called the **Interaction_Interface.** It offers two methods say **getStundentRecord** and **setStuendetRecord.** So other components just need to know these two methods to communicate with Student object.
- A component if often considered as black, the internal design description and implementation of that component is hidden from other component.
- One component can communicate with other components by several possible ways by applying suitable communication pattern.
- There are several communication patterns which can be applied while designing the communication. Patterns are nothing but well practised design strategy applied on similar kind of problems.
- We explore few architectural and communication pattern in subsequent section of this chapter. The major design patterns will be covered in next chapter.
- A component design strategy is often called the sequential design strategy where components are represented by class and component instances are represented by objects.
- Components are so independent and isolated that it often considered as plug-and-play while designing. It can assumed at the initial stage that certain kind of component can be compiled separately and kept in library of components. When the functionality is needed, that particular can be attached to the system.
- In distributed environment or networked application, components are active entities which can be deployed at any network node. For example there could be several components representing the clients and one component representing the server.

**Syllabus Topic**
**Object Oriented Software Architecture**

## 4.2 Object Oriented Software Architecture

- All the software architectures in this chapter are kind of object oriented software architecture.
- Object oriented software architecture requires that all the system diagram components and entities should follow the UML conventions. Also it should apply the basic concepts of object orientation in design as required.
- However it is possible to have the architectural design which less object components and less object oriented but the never the less they will considered as object

oriented software architecture as long as they follow the Unified processing and UML standards.

- Architectural stereotypes, different views of software architecture (dynamic and static) etc. are the integral part object oriented software architecture. These concepts related to object oriented software architecture design in explained in next sub-section.

### 4.2.1 Architectural Stereotypes in Object Oriented Software Architecture

**Q. What are architectural stereotypes? (4 Marks)**

- Stereotypes are unique kind of modelling elements specified by UML 2 and used to describe the nature of other modelling elements.
- Stereotypes specify the role characteristic of other modelling element. It is possible in an architectural design that one modelling elements plays multiple roles.
- For example a user can be an *external* user, a *devices* that user is operating to access the system or another *system* accessing the functionality as a user. So the stereotypes can be specified as <<external_user>> <<external_device>> or the <<other_system>>.
- Stereotypes are also used to represents architectural characteristics , for example one component or subsystem plays the role of *client* and another subsystem plays the role of *server*. In that case stereotypes are specified as <<client>> and <<server>> in the designing.
- We will keep on observing several stereotypes in examples from now onwards. Notice the << >> symbol is used to specify role or characteristic of any modelling element.

### 4.2.2 Architectural Views of Object Oriented Software Architecture

- Software architecture can be represented from different perspectives. These perspectives are called the different views of software architecture. These views use different modelling constructs and modelling elements to represent the design in pictorial form.
- For example, the static view is called structural design and represented by class diagrams. The behavioural view is called the dynamic view of software architecture and is represented by communication, collaboration or sequence diagrams.
- Similarly the deployment view is represented by deployment diagrams. We will start out discussion with all these view by taking an example or case study of medical centre application or medical centre system.
- All our subsequent discussion in this chapter will be based on this case study. The use case scenario and requirements are described in next section:

### 4.2.2(A) Use Case : Medical Centre System

**Case Study for Client Server Architecture**
**Medical Centre Application**

There is a medical centre in college campus serving as small hospital for the college employees and students. A system needs to be developed that automates the daily routine tasks of medical centre. We can call it as Medical Centre System or Medical Centre Application. This should be a web based application so that patients can access it remotely. Patient has to register the personal information for the first time and will get an access card which can be used for all subsequent interaction with the system. A unique patient Id will be generated along with access card.

Patient can also visit the hospital, contact the help desk and get the personal information registered in the system. Patient can fix an appointment, check the schedule, access the lab report using the online portal or can access the medical centre application system from inside the hospital. Doctor consultations can be queued and managed by the system using tokens. Doctor's assistant can type the prescription and save in system. Doctor can access the patient's history. Pathologist and lab assistants can access the prescription and save the report. Patients can upload the previous reports and download the report from the system.

The system specified in this case study, can be designed by applying an architectural design pattern. We will discuss the architectural design pattern later in this chapter in detail. The entire system can be divided into two sub-systems. One is the light weighted **patient subsystem** and another subsystem contains all the core medical functionalities called **medical service sub-system**. Patients and doctors mostly access the patient subsystem. **Patient subsystem** requests **medical service subsystem** for various needs. **Medical service subsystem** contains the central database for the system and hosts the processing logic and algorithms. The entire system can be viewed from different perspective. In most of the cases the architectural design starts with static view of the system called the structural design. Then dynamic view is captured and at last the deployment view. We will discuss these views one by one.

### 4.2.2(B) Structural View of Software Architecture

**Q. Describe the static structural view with class diagram. (5 Marks)**

- The structural view of software system is the static view and highlights the overall architecture of system.
- It represents the overall architecture in the form of different organized subsystems connected with each other.
- This view can be represented by UML subsystem class diagrams and associations between them.
- The structural view covers only those aspects which do not change with time. It is the most stable view and all subsequent views are usually designed based on this static view.
- Composite and aggregate classes are used to design the subsystems and their relationship can be depicted by associations and their multiplicity.
- Now we will carry on our discussion based on the case study specified in earlier section.
- A structured view of the medical centre application is shown in Fig. 4.2.1. A class diagram is used to depict the subsystems.
- The medical centre system is divided into a patient subsystem and a medical service subsystem. Stereotypes <<client>> <<service>> are used to characterize the system.
- Patient subsystem is a light-weighted client contains the minimal functionality and accessed directly by the user. The role of user can be played by patient, doctor, pathologist etc.
- Patient subsystem request for the services from medical service subsystem that contains the core medical functionality and patient's central database repository.
- Many patients can access the medical centre application in parallel. A user can swipe the card and check the personal information through the client system installed in hospital.
- Another at the same can access the client subsystem online and doctor can write the prescription patient. So there are multiple instances of client sub-system or the patient subsystem.
- There is single instance of **medical service subsystem** that contains all the data processing capabilities and data repository.
- In one use case scenario, a patient can register himself using the patient subsystem. The patient's information is temporarily stored in client sub system. Once approved by system administrator, it is ultimately submitted to medical service sub-system.
- The structured view of software architecture also specifies the associations and multiplicity among the subsystems.
- For example, there could be multiple instances of patient subsystem getting service from single instance of medical service subsystem. So there is one-to-many association between **medical service subsystem** and **patient subsystem.**

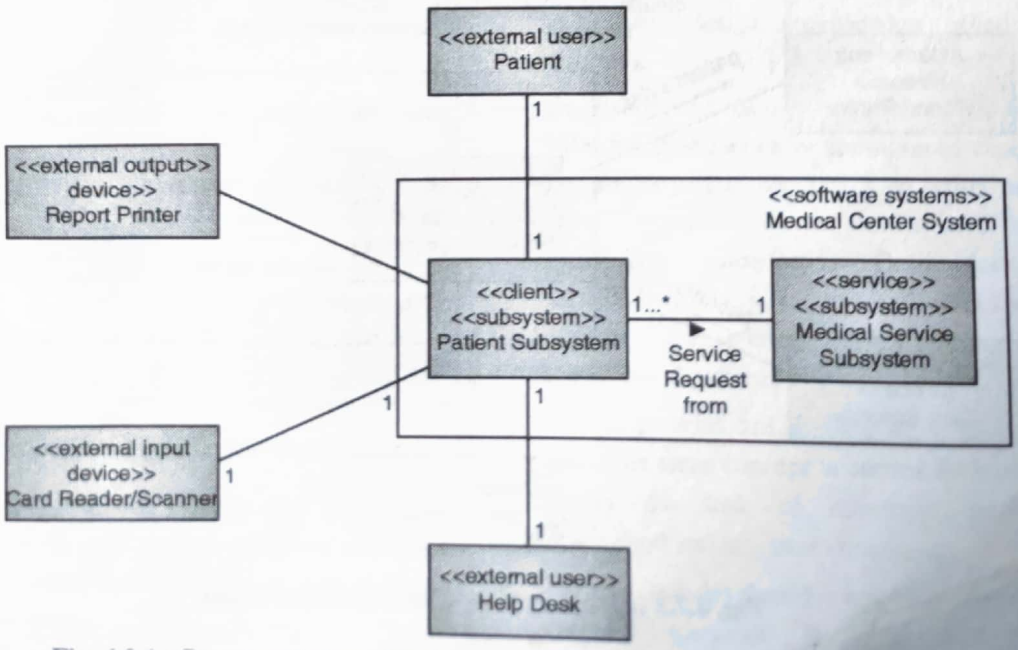


**Fig. 4.2.1 : Structural view of Software Architecture/Architectural Design**

## 4.2.2(C) Dynamic View of Software Architecture

**Q. With example, explain dynamic view of software architecture. (5 Marks)**

- The dynamic behaviour of system is captured by dynamic view of software architecture. This can be represented by a communication diagram.
- A subsystem communication diagram is used to depict the system as composite or aggregate objects along with message passing between them.
- This view specifies in abstract that how the system will behave at run time. The emphasis is on communication between object and their message passing sequences.
- Architect can choose up to what level of detail they want to capture the communication between sub systems. Generally the abstract communication is shown using the suitable communication pattern.
- A communication diagram for medical centre application is shown in Fig. 4.2.2 that captures the dynamic view of the system.
- All the possible communication in brief is shown in this diagram to represent the behaviour of system. Most of the communication here is synchronous communication where message has predefined ordering.
- Patient subsystem requests for several type of services to the medical service subsystem. For example, once registration form is submitted and system admin approves it, patient subsystem requests medical service subsystem to save the record in central database and generate the Patient Id along with access card. There are several other possible communication shown in Fig. 4.2.2.
- In Fig. 4.2.2, the synchronous message request is shown by a solid black arrowhead and reply is depicted by dotted arrow.
- Optionally message sequence number can also be mentioned along with arrows to specify the ordering of messages.
- It should be noted that dynamic view specifies the how part of the system. It means this view is more concerned about specifying how system should work in brief. But the structural view is concerned about specifying *what* system we want to develop.

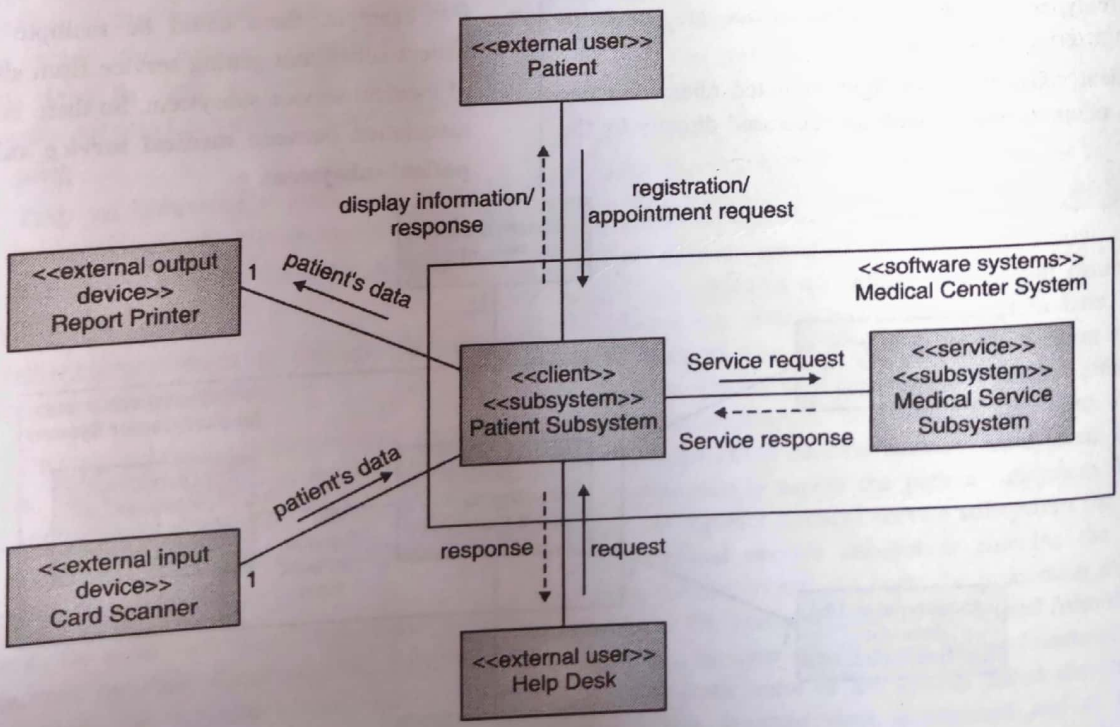


**Fig. 4.2.2 : Dynamic view of architectural design**

### 4.2.2(D) Deployment View of Software Architecture

- Another important view is the deployment view that captures system distribution in a networked environment.
- Today most of the system is web or enterprise applications and works as distributed systems. Deployment view specifies the location of each sub-system and their communication media.
- Deployment view can be depicted as deployment diagrams. It shows the deployment of subsystems at different network nodes and the type of network.
- Deployment views are more concerned about physical layout of software system. The idea is to capture the scope of system in terms of installation or deployment.
- In general it is straight forward block diagrams of subsystems and their communication media. However architects can choose to add distributed configuration details in it. For example: type of communication media, interfaces to the subsystems and protocols being used etc.
- Deployment diagram for medical centre application is shown in Fig. 4.2.3. Here several client subsystems are possible which can be deployed on different nodes within hospital premises and laboratory.

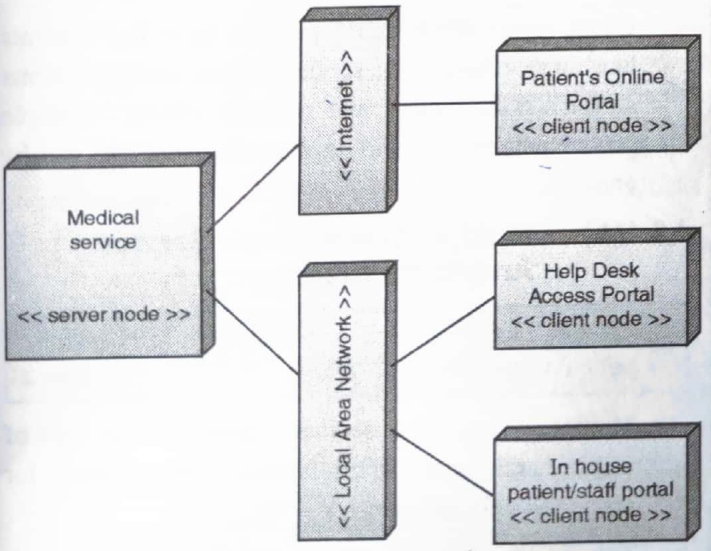


**Fig. 4.2.3 : Deployment view of Architectural design**

- Fig. 4.2.3 shows that the sub systems of medical centre application. There is one instance of server called the medical service subsystem deployed at central server called the server node.
- There are several instances of clients deployed at different client nodes. For example the web portal is hosted at one client node that can be accesses remotely by the patients through Internet.
- Inside medical centre, there is dedicated client subsystem that can read patients access card and send request to server.
- There is internal layout of local area network to connect different subsystems.

## 4.3 Software Architectural Pattern

**Q. What is software architectural pattern? And why they are important in modern software development? Explain in brief. (6 Marks)**

- In modern days of software development, designing everything from scratch is resource consuming. If past experience is not used, the designing and implementation tasks will be cumbersome.
- Over the years of experience in software engineering and architectural design, it has been realized that same kind of problems can be solved using the common design strategy.
- Architects working for decades in different kind of software projects have identified some patterns in software systems. It is possible that context and application of software system is different but at the same time it is possible that same design strategy can be applied on different projects.
- The design strategy is nothing but a well defined architectural/design specification called the **design patterns.**
- Bushmann and Gamma in their published works in 1996 has given an early definition of design pattern as "A design pattern describe a recurring problem to be solved, a solution to the problem and the context in which that solution works". Design patterns are often termed as micro architecture and when the patterns are applied at higher level software design then called the architectural pattern.
- Design patterns are applied while designing as well while the same concept is carried forward in coding. It makes the task of developer more organized, disciplined and less time consuming.
- Patterns are developed over the years of failure and success experience. So applying a pattern while

designing and implementing the systems largely minimized the risk of failure and contributes in quality of software development.

- We will discuss few popular architectural patterns in this chapter and will even more interesting design patterns in next chapter. Next section will describe the client-server architectural pattern where we take the example of Medical Centre case study.

## Syllabus Topic : Client Server Architecture

### 4.3.1 Client Server Software Architecture

→ (Feb. 2016)

**Q.** Explain client server architectural pattern.
SPPU - Feb. 2016 (In sem.), 3 Marks

- One of the simplest and widely applied architectural pattern is client-server architectural pattern. A client is the requestor of service and server is the provider of service.
- A service can be processing service, database service, file service, line printing service. Term server and service are not the same.
- Service is the application specific functionality that client is requesting for. Server is the combination of generic hardware and software that can host service.
- A server can serve the clients sequentially or concurrently. In sequential server, serve only one request at a time and other clients has to wait. In concurrent server multiple clients can be served at same time.
- Often multithreading is applied on server to create multiple threads to handle the request of each client. After the request is served, threads are destroyed or returned to the thread pool.
- We already have explored the example of client-server design pattern where medical centre system is distributed into patient subsystems and medical service subsystem.
- Client server design looks straight forward designing process but requires some crucial design decisions to be made. For example, communication mechanism, types of services and servers etc.
- We will list few of the important design decisions to be made while starting client-server architectural design pattern :
  - How many instances clients can be served by the server. Whether it is single client system or multiple client system. (Multiplicity of association)?
  - Whether the server is capable of serving concurrent requests or it handles the request sequentially?
  - Whether server is required to serve all kind of requests or client is capable of handling few requests by itself (client is thin or not)? For example, in our design of medical centre application, it is not required that patient subsystem will send request every time to medical service subsystem. Fixing an appointment and checking the doctors scheduled can be handled by patient subsystem itself.
  - Whether the communication is synchronous, asynchronous or both? In synchronous communication clients waits for request to be served. In asynchronous communication client sends the request and continues with operations.
  - Whether a single server is capable of providing the service or there is need of multiple services hosted on same or different servers?
- Client server architectural patterns have few variants like single-client single service pattern, multiple clients single service pattern and multiple clients multiple service pattern. We will explore these variants one by one.

### 4.3.1(A) Multiple Client Single Service Architectural Pattern

**Q.** Explain multiple client single service architectural pattern with a suitable example. **(8 Marks)**

- Multiple client single service is most common type of architectural pattern where multiple clients request for service to a single server.
- This is the simplest pattern where single server is serving multiple clients and so the pattern is also called client/server pattern or the client/service pattern.
- In its simplest form, single client or very few clients access the services from server. An example is shown in Fig. 4.3.1.

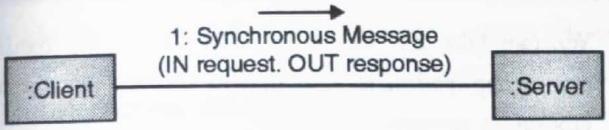


**Fig. 4.3.1 : A simple client server architectural pattern using communication diagram**

- Fig. 4.3.1 shows the client-server architecture in its simplest form. But today most of the software systems require to support multiple clients. For example Medical Centre Application has two subsystems.
- One is server that hosts the medical service subsystems and other are clients hosted by different nodes in the hospital premises.
- Fig. 4.3.2 shows the multiple clients single service client-server architectural pattern in its true sense. It shows two clients instances of client sub system interacting with single medical service sub system.
- In modern web applications and networked applications, the number of clients is not limited to one or two. Web applications are designed to handle thousands of requests simultaneously.
- In the case of study of medical centre application several patient can access the hospital service. It means several instances of patients subsystems will be accessing the medical service subsystem.
- For example patient will be accessing the online portal of medical centre to perform various activities remotely like patient registration, fixing the appointment, accessing the report.
- At the same time , one patient can access the hospital's patient subsystem using the access card. Doctor can also access the patient subsystem to check the scheduled appointments. Help desk can access the system to fulfil the request of registration for new patient and grant access card.
- A more detailed multiple-client single service architectural pattern example is shown in Fig. 4.3.3.

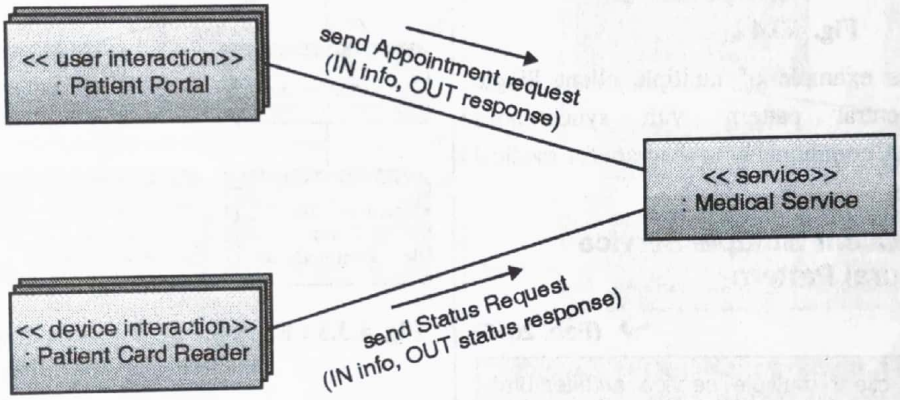


**Fig. 4.3.2 : Multiple client single service architecture example : A communication diagram for Medical Centre Application**

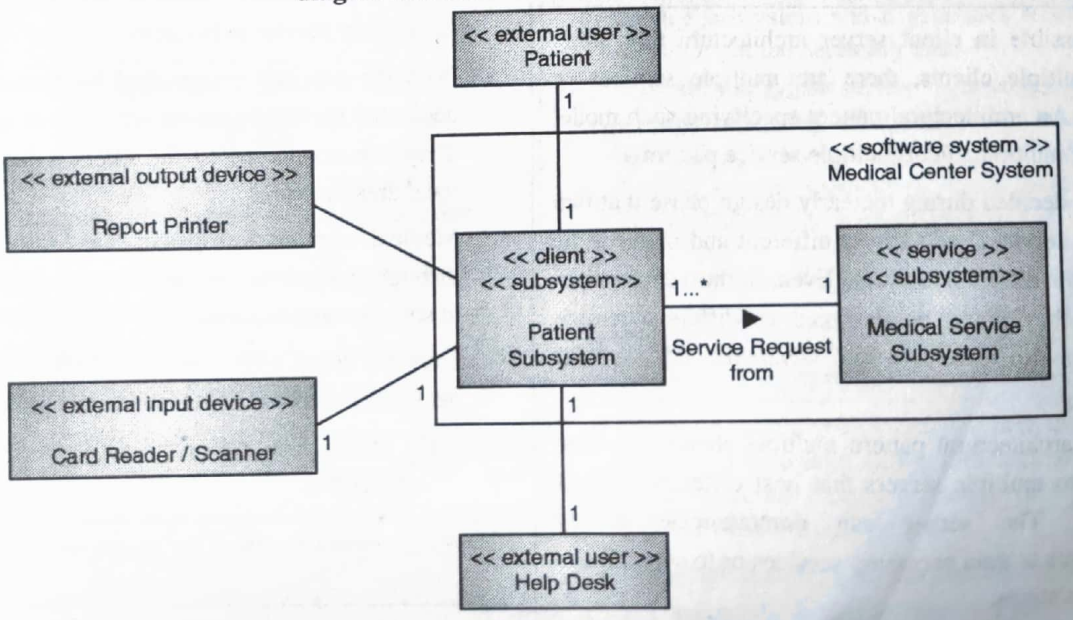


**Fig. 4.3.3 : A structural view of multiple client-single service architectural pattern**

- There could be multiple model of communication is possible between client and server. In possible method, a client sends a request to the server and waits for the response. This is called synchronous communication.
- For example a patient subsystem can send the request to server (medical service subsystem) for retrieving patient's history of treatment. Once the patient subsystem sends the request, it waits for reply from server.
- Similarly there could be several instances of patient subsystem requesting some service from server in a synchronous model. The scenario is depicted in Fig. 4.3.4.

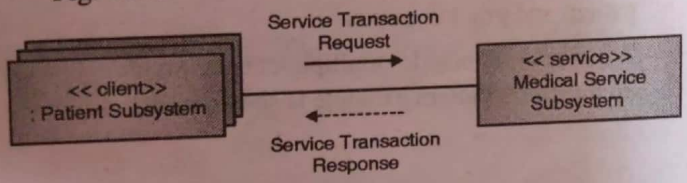


**Fig. 4.3.4**

- Fig. 4.3.4 shows example of multiple client Single service architectural pattern with synchronous communication: A communication diagram for medical service subsystem.

### 4.3.1(B) Multiple Client Multiple Service Architectural Pattern

➔ **(Feb. 2016)**

**Q.** Explain multiple client multiple service architectural pattern with a suitable example.
**SPPU - Feb. 2016(In sem), 2 Marks**

- It is possible in client server architecture that along with multiple clients, there are multiple services or servers. An architectural pattern specifying such model is called multiple client multiple service patterns.
- It often decided during the early design phase that two or more services are entirely different and better to be hosted by different servers. Even if the services are related, they could be designed as different service subsystem to leverage the advantage of modular designing.
- In such architectural pattern multiple clients can send request to multiple servers that host different kind of services. The server can communicate among themselves to gain necessary services or to synchronise their operations.
- We can take the where requirements of the medical centre application is extended to include some major laboratory services.
- System architects have now decided to logically and physically separate the laboratory service subsystem from the core functionalities of medical service subsystems.
- Medical service subsystems will mainly get accessed by patients, doctors, help desk etc.
- Laboratory services are dedicated for storing and analysing different kind of test results and mainly used by pathologists and lab assistants. Fig.4.3.5 shows deployment diagram for multiple client-multiple service architectural pattern.

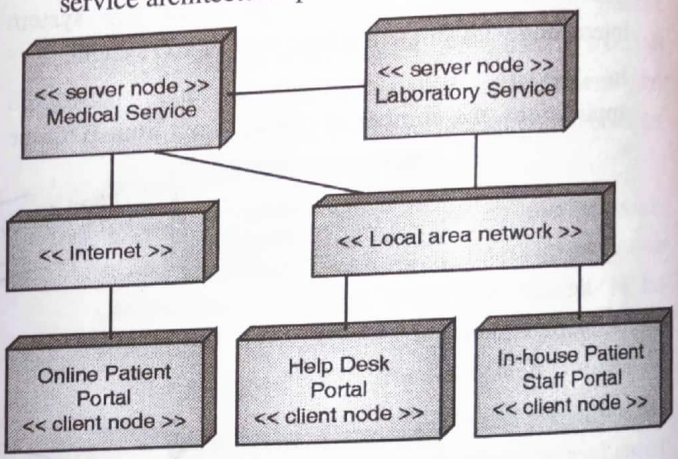


**Fig. 4.3.5 : Example deployment diagram for Extended Medical Service Subsystem**

- Fig. 4.3.5 shows the entire service distribution into two subsystems namely Medical service subsystem and Laboratory service subsystem.
- Both the services designed to be hosted on different dedicated servers in the server room in medical centre. They are connected to the client subsystem through local area network.
- Medical service subsystem can also be accessed through Internet in case online patient portal is hosted at some remote location.
- Even the client's browser can act like a thin client and get the services from Medical service subsystem.

### 4.3.1(C) Multi-tier Client-Service Architectural Pattern

**Q.** Explain multi-tier client server architectural pattern. **(4 Marks)**

- Multi-tier client-service architectural pattern places one or more extra layer of service between conventional client and service.
- The intermediate tiers take the services from upper layer and hence acts like a client for the upper layer services. At the same time it can provide to services to lower level sub-systems and acts like a server.
- Service offers the well defined interface to its clients. Client can request the server using the offered interface.
- A client only needs to know about the interface and service offered by its immediate above layer. So the upper layer can also send the request to its upper layer to fulfil the request without its client's awareness.
- There could be several layers of services but increasing the layer of services in design can make design unclear and ambiguous. So the services should be carefully broken into layers.
- It should be made sure that breaking the service multi-tier layer architecture will positively affect the implementation process.
- One abstract layering of services is shown in Fig. 4.3.6 where one layer of database service is added in medical centre application.

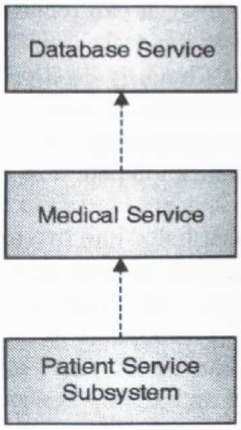


**Fig. 4.3.6 : Example of multi-tier client server architectural pattern: Medical centre application**

- There could be a dedicated database server that hosts all databases and well defined interface to the medical service to access those data. So **medical service subsystem** becomes the client for data base service.
- **Patient subsystem** request for different services to the **medical service subsystem** which hold all the data processing capabilities and algorithmic capabilities but cannot store the data permanently. For storage and retrieval of data, it depends on the services provided by database server.
- A deployment diagram for multi-tier client service architectural pattern is shown in Fig. 4.3.7. The layered services can be observed here where layer 3 is the top layered service.
- The detailed deployment diagram of medical centre application is used here to depict the multi-tier client-service architectural pattern.
- Instances of patient subsystems at layer 1 can generate the request for service hosted at layer 2. For example can use one instance of patient subsystems at layer 1 to request a patients all previous medical record in ascending order.

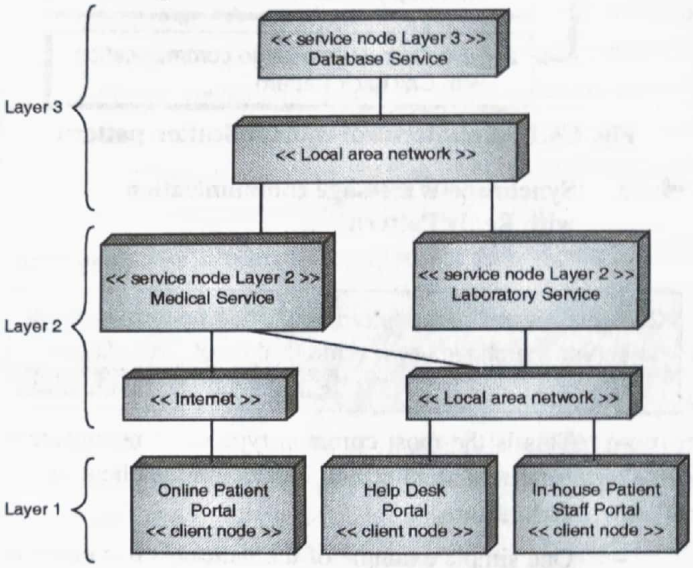


**Fig. 4.3.7 : Deployment diagram for multi tier client service architectural pattern**

- There request is interpreted at layer 2 by **medical service subsystem** and it generates request **data base server** to get the necessary data.
- Once the immediate service layer which is the **medical service subsystem** obtain the data from the above layer, it processes data if required and sends processed data to patient subsystem. Then patient subsystem displays the data in required format.

## 4.3.1(D) Architectural Communication Pattern

**Q. What is important communication patterns applied in client-server architecture? (6 Marks)**

- There are certain communication patterns also that can be applied in design along with architectural pattern. These patterns define a model of communication between different subsystems.
- The simplest pattern is where one client sends a request and waits idle for reply. Server receives the request, processes the request and responds accordingly.

- Once the response is received by client, it resumes its operation. This is simple synchronous method of communication.
- There are few cases where response is of minimal significance. Like update request at server, where client just need an acknowledgement that the request operation has performed successfully.

☞ **Two architectural communication pattern**

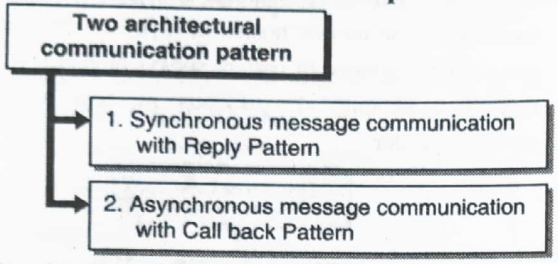


**Fig. C4.1 : Architectural communication pattern**

### ➜ 1. Synchronous message communication with Reply Pattern

➜ **(May 2016)**

**Q.** Explain synchronous communication pattern in client-server architecture with help of a diagram. **SPPU - May 2016, 5 Marks**

- This is the most common type of communication pattern in architectural design used in client server architecture.
- One simple example of the pattern is one client is requesting for service to server and waiting for reply. There is no message queue developed between that client and server and once server responds client proceeds with its operation. Fig. 4.3.8 shows such communication between client instance and a service instance.

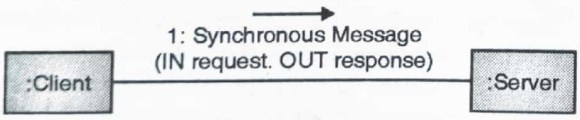


**Fig. 4.3.8 : Simple One client and one server setup : Synchronous message communication with Reply Pattern**

- Commonly there are more than one client requesting for the service and then waits for the response.
- Fig. 4.3.9 shows the same pattern with two instances of clients. One is to be deployed as Patients portal which patients can access online. Another is Patient subsystem installed with card reader within the premises of medical centre.
- Both clients sub systems and medical service subsystems implements the Reply Pattern of synchronous communication.

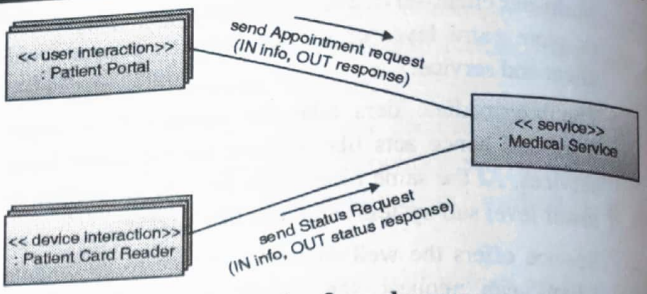


**Fig. 4.3.9 : Example of synchronous message communication with Reply pattern : Medical Centre Application**

- In such architecture of communication, it possible that message queue is build up at server because of several requests from different clients. It is the matter of internal implementation of server that how it handles multiple requests.

### ➜ 2. Asynchronous message communication with Call back Pattern

**Q.** Explain asynchronous message communication with call back pattern with suitable example. **(4 Marks)**

- Asynchronous message communication with call back pattern is applied between a client and server communication where client can send a request to the server and continues with its normal operation.
- Client does not need to wait for servers reply. But at later any time, it can receive the response and perform the related task.
- The term **call-back** is used to refer the response that client receives as a response to the request sent by client previously. One practical example of such communication pattern is the student's online examination system where the entire **examination system** is distributed into two subsystems.
- One is client that is **student examination portal** and capable of runtime question series management. Another is *examination server* that hosts question series and results. Students can view, select and submit the answer by accessing examination portal.
- On each answer submission, **student examination portal** sends the request for submission to server but does not wait for response or acknowledgement, instead displays the next question to the student.
- However the collective response can be received as acknowledgment at any later stage. In the scenario describes above, the communication pattern applied is **asynchronous call-back pattern.** Fig. 4.3.10 shows this architectural communication pattern.
- It should be noted that despite of asynchronous execution by client, the pattern follows the client/server paradigm where client sends only one request at a time and receives a response from the server system.

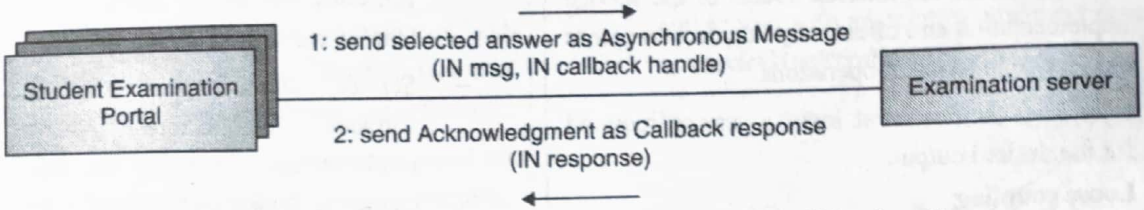


**Fig. 4.3.10 : Asynchronous message communication with call-back pattern: Example of Online Examination System**

**Syllabus Topic : Service Oriented Architecture**

## 4.3.2 Service Oriented Architecture

**Q. What is service oriented architecture? (4 Marks)**

- Client-Server architectural pattern has been explored in last section. Another architectural pattern is Service Oriented Architecture often termed is SOA. It is popular in modern days distributed web applications.
- A service itself can be defined as functionality which is very well-defined and self-contained. It does not depend on the state of other service or on the context of other services.
- Service oriented architecture is distributed software architecture and consists of multiple autonomous services.
- This is the extensions component based and client-server architectural pattern where one application components provides service to another component using the standard communication protocol over the network.
- The motivation behind such architectural pattern is that the services can be provided by different vendors and different services may use different technologies for its internal implementation.
- Services can communicate with each other and can exchange information using the standard protocols.
- SOA also defines the mechanism for allowing the applications to discover the services and communicate with it. For the purpose, there is service description associated with each service.
- Service description specifies the name and address or location of the service. It also defines the standard interface to communicate with the service.
- A service can be used by multiple clients. A client has to register for the service that it wants to access.
- Web services is the technology often used as the connection technology in a service oriented architecture.
- The Open Group has provided the standard definition of SOA as "A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations"

### 4.3.2(A) Design Principle for Services

**Q. What are important design principles for service oriented architecture? (4 Marks)**

SOA encourages few design principles to be adopted during the service design. These are few best practices software architect and engineers uses while developing the software system and equally desirable in service design. The design principles are listed in Fig. C4.2.

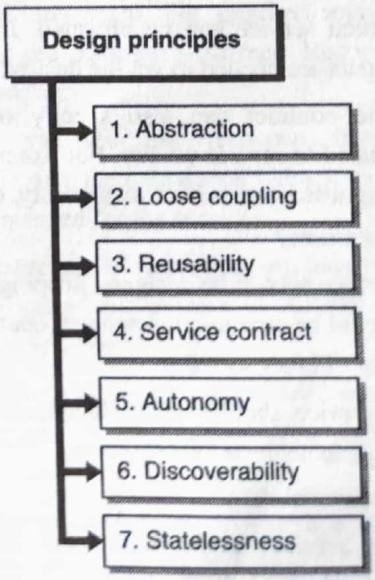


**Fig. C4.2 : Design Principles**

➜ **1. Abstraction**

- This is the primary motivation in object oriented designing where the complexity is hidden from service getters.

- A service hides the internal details of the service implementation and offers only interface to access the service in terms of operations.
- Interface specifies what input a service required for the desired output.

➜ **2. Loose coupling**

- The services should be independent of each other. In other words services should be self contained.
- It should be designed in such a way that a service should not depend on other services for its operation. Ideally a service should not communicate with other service while serving the request.
- But in many practical cases it is not possible achieve the ideal case but architects tries minimize the inter service communication.

➜ **3. Reusability**

- One of the primary goal of service oriented architectural pattern is to design reusable service.
- A service should be designed such that can be reused among different clients.

➜ **4. Service contract**

- A service contract specifies a well defined service interface that service consumer should be aware of.
- This contract is the specification of operations by which service can be obtained. It specifies what inputs are needed to get the desired output.
- The contract can also specify other parameter related to service quality. For example, the service response time, service availability, etc.

➜ **5. Autonomy**

- Service should be designed keeping in mind that it should be autonomous and can operate without the help of other services.
- A service should be self contained and hence it helps to achieve loose coupling.

➜ **6. Discoverability**

- The service should implement and support an external mechanism by which service can be discovered.
- It should be able to define the service discovery.

➜ **7. Statelessness**

- Ideally a service should not depend on client's state or should not be aware of specific client's activities. For service all the consumers should be similar and should remain in same state.
- Service oriented architecture itself uses few software architecture patterns for its implementation. One of the important patterns called the Broker pattern used to implement SOA.

## 4.3.2(B) A Software Architectural Pattern for SOA : Broker Patterns

➜ **(Feb. 2016, Dec. 2016)**

**Q.** Explain location transparency and platform transparency in service oriented architecture. **SPPU - Feb. 2016 (In sem), 5 Marks**

**Q.** Explain the Broker Patterns for design of service oriented architecture. **SPPU - Dec. 2016, 5 Marks**

- In Service Oriented Architecture, there is an intermediary between client and the server called broker.
- Brokers hide all the complexities related to obtaining the service from the client. So clients do not have to keep and maintain information about where and how to get the service. Broker helps client to locate and obtain the service.
- Broker is often termed as Object Broker. And this pattern of getting the service is called simply as **Broker Pattern** or as the **Object Request Broker Pattern.**

☞ **Two important facilities provided by the broker**

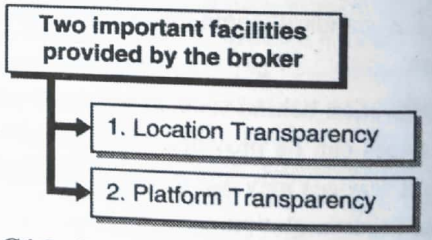


**Fig. C4.3 : Two important facilities provided by the broker**

➜ **1. Location Transparency**

- The location transparency specifies that irrespective of the change in location, client should keep on getting the same service.
- If the service moves from location to another, client does not have to make any configuration change.
- Only broker should get the notification of changed location. Broker should take necessary action to make sure that client remains unaware about service location change.

➜ **2. Platform Transparency**

- Clients do not need to know the hardware and software details of the services. Each service can be executed on a different hardware and software.
- Client does not need to customize itself based on service platform and does not need to maintain information about service platform. Broker handles the details of the service platform.

In broker pattern, broker has the responsibility of service registration. So instead of directly accessing the service, client consults the broker to get the information about the service.

There is special dedicated pattern for service to get registered with broker called the **Service Registration Pattern**. There are certain sub patterns which can be opted while designing service oriented architecture.

☞ **Sub patterns**

The list of the sub patterns is provided in Fig. C4.4.

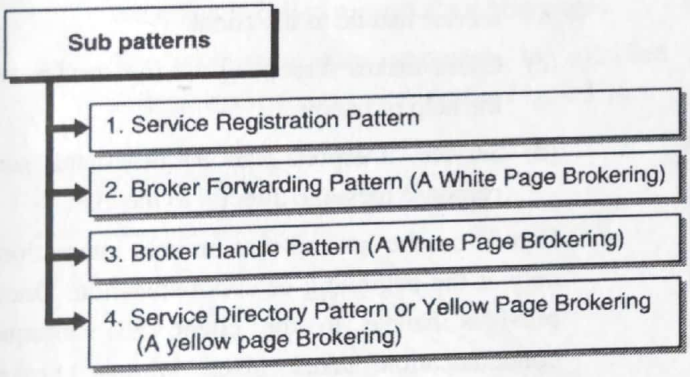


**Fig. C4.4 : Sub patterns**

➜ **1. Service Registration Pattern**

- Service has to register itself with the broker. Here service provides all the information to broker. This information primarily includes:
  1. Service Name
  2. Service Description
  3. Service Location
- Service needs to register itself only once. Reregistration is required when information changes. For example when service moves from one location to another.
- A simple scenario of service registration is shown in Fig. 4.3.11.
- Here one service say S1 sends a request message named **serviceRegistrationRequest** to a broker B1.
- The broker registers the service in service registry and replies with an acknowledgement named here as **serviceRegistrationAck.**

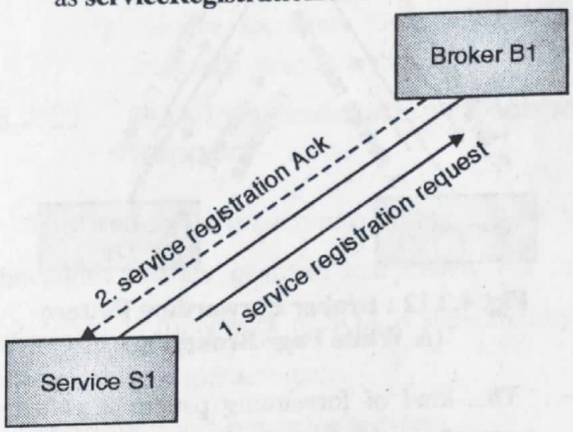


**Fig. 4.3.11 : Service registration Pattern**

➜ **2. Broker Forwarding Pattern**

- There are several ways by which a broker can handle the client's request. One is specified by Broker forwarding pattern.
- Clients first determine which service required for its operation. It forms a request message using agreed protocol and in required format, sends it to broker.
- Broker analyses the requested services, locates the service and simply forward the request to service provider.
- The message is received at the service provider's site and interpreted. The required service is then invoked and response is formed. Response is sent to the broker.
- Broker receives the response from service provider and then forwards it back to the client who requested for the service.
- The sequence of messages exchanged between client, broker and service is summarized below and shown in Fig. 4.3.12.
  (a) Service requestor or the client requests for the service by sending request message to the broker.
  (b) Broker locates the service and forwards the service to service provider.
  (c) Service provides interpret the request and sends a response message to the broker.
  (d) Broker forwards the response message to the client.

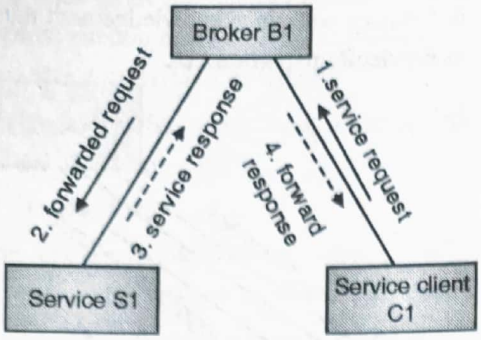


**Fig. 4.3.12 : Broker Forwarding Pattern (A White Page Brokering)**

- This kind of forwarding pattern is called **white page brokering** because client knows the service it requires but does not aware of the location of the service. This is somewhat similar to white pages in telephone directory.
- In broker forwarding pattern every message sent between client and server, passes through broker. It provides an advantage of carefully examining each message before forwarding at broker side and hence enhancing the security.
- High security comes at the cost one extra level of redirection of message.
- There is no direct communication between client and server. This degrades the network performance.

➜ **3. Broker Handle Pattern**

- In broker handle pattern, broker initiates the communication and then facilitates a direct communication between client and service.
- Broker provides a service handle to the client, which is used for direct communication between client and server.
- This way, network performance is increased by removing extra level of redirection of messages. Location transparency is maintained as the handle provided by broker to client at beginning of communication takes care of service location.
- This pattern works well when there is heavy communication between client and service and we also want to keep the message traffic as low as possible.
- Since the location transparency is maintained, this pattern also falls in the category of white page brokering.
- The communication sequence is listed below and shown in Fig. 4.3.13.

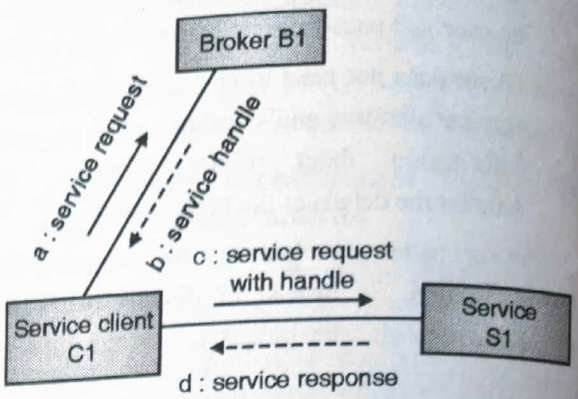


**Fig. 4.3.13 : Broker Handle Pattern (A white page brokering)**

(a) Service requestor or the client requests for the service by sending request message to the broker.

(b) Broker locates the service and returns a service handle to the client.

(c) Client makes direct request the service with the help of handle.

(d) Service interprets the request and sends response message directly to the client.

- Here it can be noticed that brokers interaction is only at the beginning of communication. Once it provides handle to the client, all subsequent communication takes place without broker's interaction.
- This is more efficient than Broker Forwarding pattern as redirection of messages through broker is significantly reduced.
- Once the communication session is over, the handle is of no use and client is expected to discard it.

➜ **4. Service Discovery Pattern**

- The previously discussed two patterns were white page brokering pattern where client is completely unaware about the location of the service.
- In this type of pattern, client knows what service is required but does not know where that service is located.
- Another type of brokering is yellow page brokering .It is somewhat similar to yellow pages of telephone directory. Client has very less

knowledge about the service. It knows the type of service but not the specific service.

- This kind of pattern called the Service Discovery Pattern, allows the various clients to discover the new available services.
- Client first specifies the type of service it wants. Broker provides the list of available services to the client by an initial response message. Then client selects the specific service that could be based user preference. Client sends it to selection or preference to broker. Broker responds with a handle to the selected service. Using the handle client can communicate with service.
- Once client selects the service and requests to the broker, the communication pattern becomes similar to white page brokering.
- That is why the first request by the client is called request for **yellow pages** and second request by the client is called request for **white pages.**
- The communication sequence is specified s follows and pattern is depicted in Fig. 4.3.14.

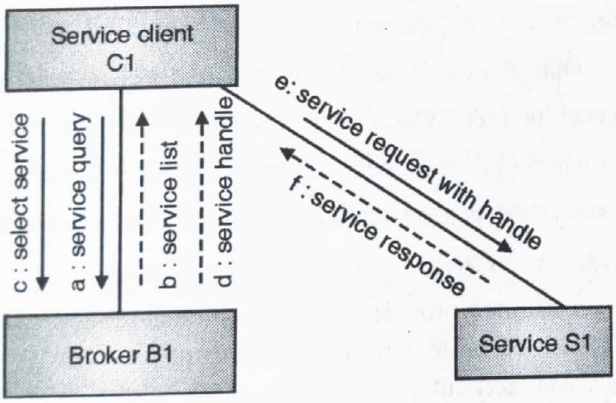


**Fig. 4.3.14 : Service Discovery Pattern (A yellow page brokering)**

(a) A yellow page request is generated by client i.e. the request for all the services of the type specified by client.

(b) Broker responds by providing the list of all services matching the type specified by client in request.

(c) Client selects the service and generates a white page request for the selected service to the broker.

(d) Broker locates the service and responds by sending the handle for the requested service.

(e) Client generates the request by using the handle.

(f) Service interprets the client's request and responds directly to the client.

## 4.3.2(C) SOA Implementation : Technology Support

Services can be implemented using a wide range of technologies and are platform independent but the entire implementation framework of SOA is supported by special services called the web services.

☞ **Important implementation services**

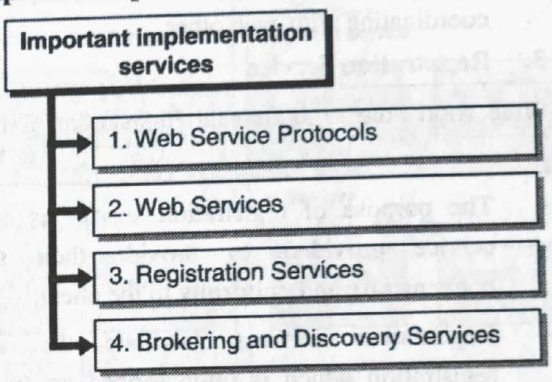


**Fig. C4.5 : Important implementation services**

### ➜ 1. Web Service Protocols

**Q. Write short note Web Service protocol. (3 Marks)**

- Web service protocols provide a standard for communication between services and clients.
- World Wide Web (Consortium) popularly known as W3C has developed a light weighted protocol called SOAP (Simple Object Access Protocol) based on XML and HTTP messages.
- It provides the foundation for communication for Service Oriented Architecture and largely used in distributed environments.
- SOAP messages consist of three components.
  (a) **Envelop** that describes the message
  (b) **Encoding Rules** for the sender, receiver and the data type description
  (c) **Convention** for representing response messages and Remote Procedure Calls

### ➜ 2. Web Services

➜ **(Feb. 2016)**

**Q. Write short note Web Service**
**SPPU - Feb. 2016 (In sem), 5 Marks**

- Web services provide a standard and convenient way for application to application communication over the Internet.
- It provides an API (Application Program Interface) as an standard way of communication between different software hosted on Internet as services.
- It is the web service that makes a software hosted on any site on Internet to be available as service. For example if an organization wants to provide certain functionalities online, they can opt for representing their functionalities as web services.
- A service can be composed of several applications coordinating with each other.

➜ **3. Registration Service**

**Q. Write short note Web Service Registration Services. (3 Marks)**

- The purpose of registration service is enabling service providers to provide their services conveniently and uniformly to the client.
- Registration services facilitate the services registration which is often termed as publishing the service.
- Registration services are mostly provided by the Object broker. For example the CORBA and Web Service Brokers provide such services.
- In web services, broker provides the service registry option to the services so that they can publish their services online and becomes discoverable.
- Clients can consult the broker and find the service registry for suitable service. WSDL (Web Service Description Language) is mostly used for describing the services.

➜ **4. Brokering and discovery services**

**Q. Write short note Brokering and discovery services. (3 Marks)**

- In service oriented architecture, one of the important role is played by object brokers that shields the complex details of service from the client.
- The activity of intermediating between client and service is called brokering. It has the primary responsibility of allowing the clients to discover the services.
- A framework called UDDI (Universal Description, Discovery and Integration) is used for the purpose.
- UDDI consists of web xml documents and XML schema enables the client to dynamically find service over the Internet.

## 4.3.2(D) A Case Study for Service Oriented Architecture : e-shopping System

➜ **(Feb. 2016)**

☞ **Problem Description**

e-shopping (also known as Internet shopping, Online shopping, e-web store etc.) is an example of modern e-commerce system that allows users, buyers or consumers to buy the goods or services online. The system is accessible online through web browser or mobile app. Today it is taking the form of m-commerce and sometimes deeply integrated with m-commerce where users access the system through mobile optimized sites or apps.

The e-shopping system is web based online system. A buyer can browse the catalog, pick the items in shopping cart, update the shopping cart, check out and make the payment. Customer/buyer can track the placed order and cancel the order. Customer can write the feedback and can submit the product review. Supplier/seller can login in the system can process the order, update the order status, update the catalog to display seasonal offers etc.

Customer provides personal information like address and card (debit /credit) details which are stored in system in users account. Once information is verified by the system, the order gets placed and system takes the necessary action to charge the customer's credit/debit account. An email is sent to the registered customer email id stating successful placement of order. Seller is notified with order details. Seller checks and verifies the order then confirms the order with expected delivery date and courier information. A mail is generated and sent to the customer as a confirmation from seller.

☞ **Use case modelling for case study : e-shopping system**

- A high level use case is illustrated for the e-shopping system where the primary actor is buyer and seller.

- We can make a formal description table for various scenarios like view and pick items, make order request, process order etc. in same way as we learnt it in Chapter 1.
- We can also design activity diagrams (as described in chapter 1) for these scenarios with suitable assumptions.
- For now will be covering the Use case diagram and proceed towards static structuring of system. The use case diagram is shown in Fig. 4.3.15.

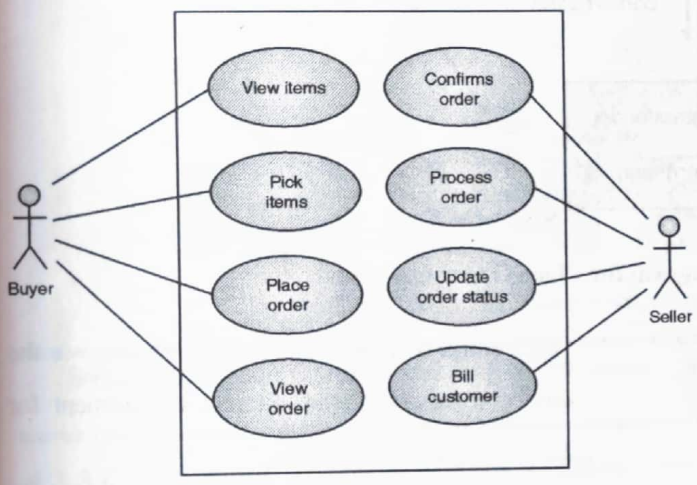


**Fig. 4.3.15 : Use Case : e-shopping system**

### Static Modeling : Class and Object structuring

- In static modeling for an application to be formed as service oriented architecture, services are designs high level classes.
- The primary services are identified here with their respective entity classes.
- Entity classes are represents the entities in application domains which has certain attributes.
- A service class provides access to entity classes. The stereotypes <<entity>> and <<service>> are used to represent and entity and service classes respectively.
- Fig. 4.3.16 shows the service and entity classes for e-shopping system. Note that there could be more services and classes that can be added in this diagram.
- For the sake of simplicity, we picked the important classes that influence the online application.
- For example Courier Service class can also be added with <<service>> stereotype which provide access to the Parcel <<entity>> class.
- Similarly Email service can be added that has dedicated task of generating emails.

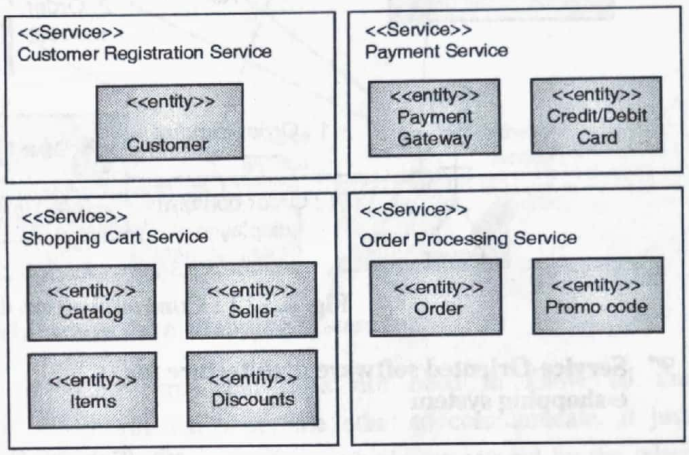


**Fig. 4.3.16 : Service and entity classes for e-shopping system**

### Dynamic Modeling for e-shopping system

- System can be dynamically modelled for the different use case scenarios.
- For example one dynamic modelling is performed to show how the customer interact with system while registering the personal information.
- Another can be shown to capture the **Catalog browsing** or **View Item** use case. Another can be shown for **Order Processing** use case.
- This can be designed with the help of communication diagrams showing messages getting passed between different services.
- One example is dynamic modeling shown in Fig. 4.3.17 that captures the use case where customer/buyer places the order.

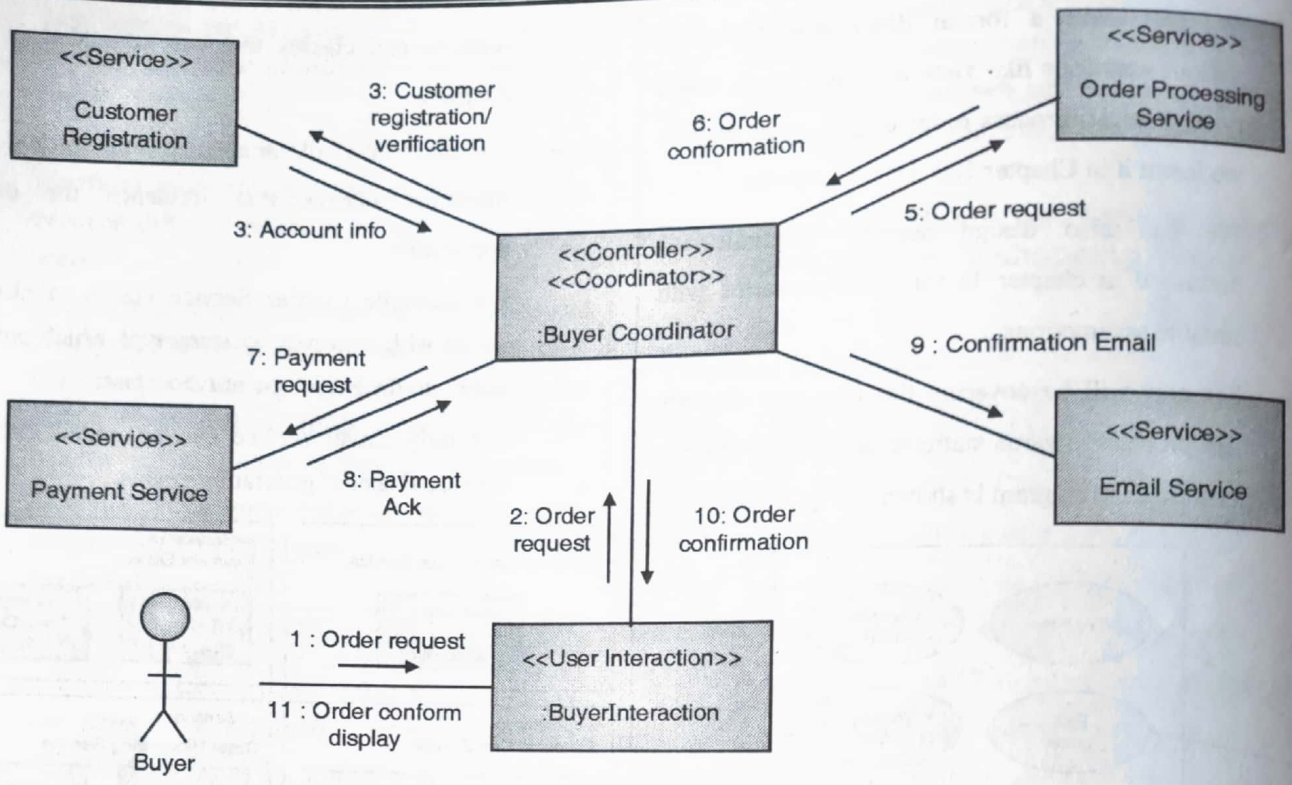


**Fig. 4.3.17 : Communication diagram for Place Order use case**

### Service-Oriented software architecture for e-shopping system

➔ **(Feb. 2016)**

**Q.** Explain Port, provided interface and required interface with an example.

**SPPU - Feb. 2016 (In sem). 5 Marks**

- Service oriented architecture for e-shopping system can be organized into three layers: Service layer, Co-ordination layer and User layer.
- Service operations are accessed by one provided interface. Clients can invoke the appropriate service provided by interface.
- Services are represented by individual components in the diagram.
- Interfaces are attached to its internal service by well defined port which is represented by small square shaped box attached to the component.
- The provided interface represents a formal contract of the services that consumer gets.
- There could be required interfaces attached to the service representing the external requirement for the service for its operation.
- For example OrderProcessingService may have one provided interface (PI) called PI_OrderProcessing and one required interface (RI) called RI_Payment Service.
- The required and provided interfaces are linked to internal service by required and provided ports respectively.
- Fig. 4.3.18 shows the layered service oriented architecture of e-shopping system.
- Ports starting with P represent provided ports and ports starting with R represent required ports.

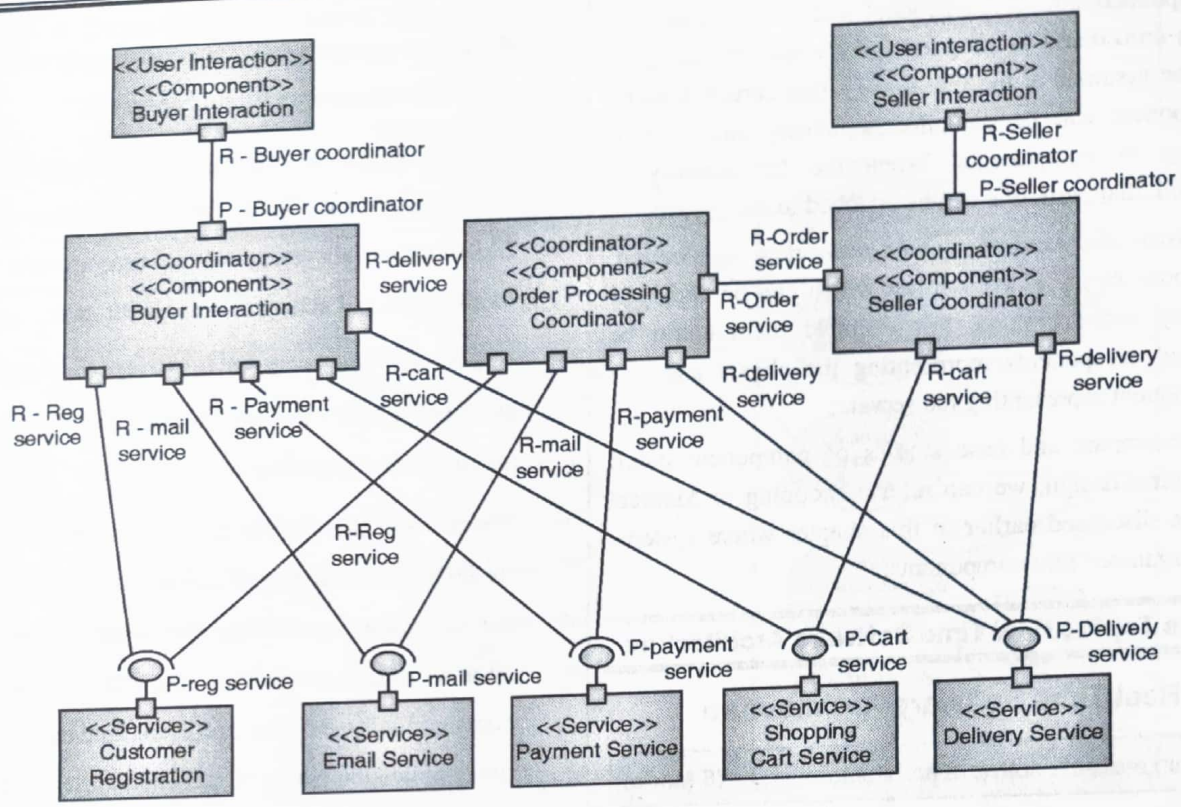


**Fig. 4.3.18 : Service Oriented Architecture for e-shopping system**

**Syllabus Topic : Component Based Software Architecture**

## 4.3.3 Component Based Software Architecture

We have covered the component based software architecture earlier in this chapter. We will revisit it here in this section and will carry forward the discussion of architectural patterns.

- One of the major concepts associated with software architecture is component based software architecture. This is the structural perspective of software design.
- In component based software architectural design, the primary focus is in dividing the system in components. Each component is well defined self contained block of system.
- A component encapsulates all the complex information about itself inside it. So a component does not describe how the system or the component works internally but it specifies what responsibility it has.
- A component can be a simple object like Student object or it can be a composite object like Department object containing other simple objects.
- A component also specifies the interface through which it communicates with other components.
- One component does not need to know all the information about the other to communicate. It just needs to know what methods are offered by the other component's interface to communicate.
- For example Student object may offer an interface called the **Interaction_Interface**. It offers two methods say **getStundentRecord** and **setStuendetRecord**. So other components just need to know these two methods to communicate with Student object.
- A component if often considered as black, the internal design description and implementation of that component is hidden from other component.
- One component can communicate with other components by several possible ways by applying suitable communication pattern.
- There are several communication patterns which can be applied while designing the communication. Patterns are nothing but well practised design strategy applied on similar kind of problems. We have already explored few architectural and communication patterns. The major design patterns will be covered in next chapter.
- A component design strategy is often called the sequential design strategy where components are represented by class and component instances are represented by objects.

- Components are so independent and isolated that it is often considered as plug-and-play while designing. It can be assumed at the initial stage that certain kind of component can be compiled separately and kept in library of components. When the functionality is needed, that particular can be attached to the system.
- In distributed environment or networked application, components are active entities which can be deployed at any network node. For example there could be several components representing the clients and one component representing the server.
- For examples and case studies of component based structural design, we can refer e-shopping or Medical centre discussed earlier in this chapter where systems are organized into components.

**Syllabus Topic : Real Time Software Architecture**

### 4.3.4 Real Time Software Architecture

**Q.** Explain real time software architecture. **(5 Marks)**

- Real time systems are special systems and has additional requirement during the design process. It has to deal multiple stream of input during its operations and has to quickly respond on real events.
- Real time software architectures are usually concurrent architectures and are state dependent. In concurrent software architecture, the system is organized or structured into concurrent tasks.
- The architecture also defines interfaces between the concurrent tasks and the interconnection between the concurrent tasks.
- Real time system architects have to carefully decide and fix the criteria regarding which tasks or subsystems should be taken as concurrent tasks. This design is crucial stage of system implementation as one wrong design decision may cause disastrous outcome in real time environment.

### 4.3.4(A) Characteristics of Real Time System

→ **(Feb. 2016)**

**Q.** Explain the important characteristics of real time software architecture.

**SPPU - Feb. 2016 (In sem), 4 Marks**

- Real time systems are concurrent systems and have strict time constraints for its operations.
- In practical, real time system is combination a real time operating system, real time application, real time Input/Output subsystem and real time specific hardware like sensors and actuators along with its drivers.
- These are widely used in healthcare and medical systems, military system, weather forecasting systems and in several industrial applications.
- The focus of this chapter is on real time application and its design. Real time application design is dependent on the hardware capabilities. For most of the real time systems, the hardware resource is limited or conventionally different desktops and servers. A real time application has to be designed accordingly.
- Fig. 4.3.19 shows typical layout of a real time system. Here we can observe that the system is dealing with multiple flow of input or output streams and few of them may needed to be handled concurrently.
- Real time systems are unpredictable and timing of data arrival is unknown. The design has to take care of certain aspects that are not considered while dealing non real time systems. Loads to the system may vary at different times which are also not predictable.

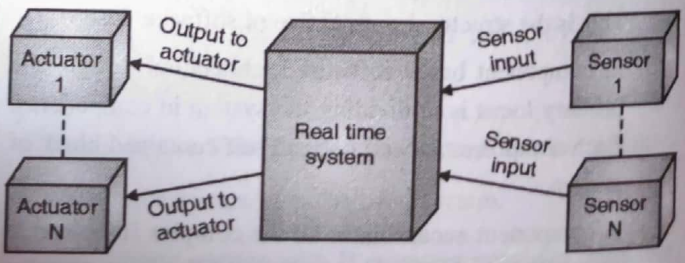


**Fig. 4.3.19 : A layout of a real time system and I/O streams**

- Real time systems have different categories that may require design strategy. For example a hard real time system may need different strategy and pattern for implementation comparing to a soft real time system.

## 4.3.4(B) A Case Study for Real Time Software Architecture : A Home Automation System

**Q.** Draw use case, class diagram and communication diagram for the Home automation system. **(10 Marks)**

☞ **Problem Description**

Home automation System is a real time system to automate the different activities of house work and house hold. It provides centralized control for electricity, appliances, security locks, fire and smokes etc. The system can be controlled locally though the control panel inside home or remotely through the mobile device. Internet of Things are tied closely to the system implementation enabling the devices acting and communicating over the Internet.

User can access the control panel at home and control devices and their configurations. User can also create the profile and set the preferences and system is able to detect the presence of registered user and can adjust the home environment as registered preference.

User can also control the devices by the remote control panel existing in the form of mobile app. User can receive the notification for different events for which user has registered in home automation controller system.

☞ **Use case modelling for case study : Home Automation System**

- Use case modeling for Home Automation system is shown in Fig. 4.3.20. The primary actor is the user who can crate user profile, set preferences and control devices.
- There are separate electricity, security and appliances sub systems which are connected with required switches, sensors and actuators.
- These subsystems externally interact with the central home automation controller system and serves as actors.

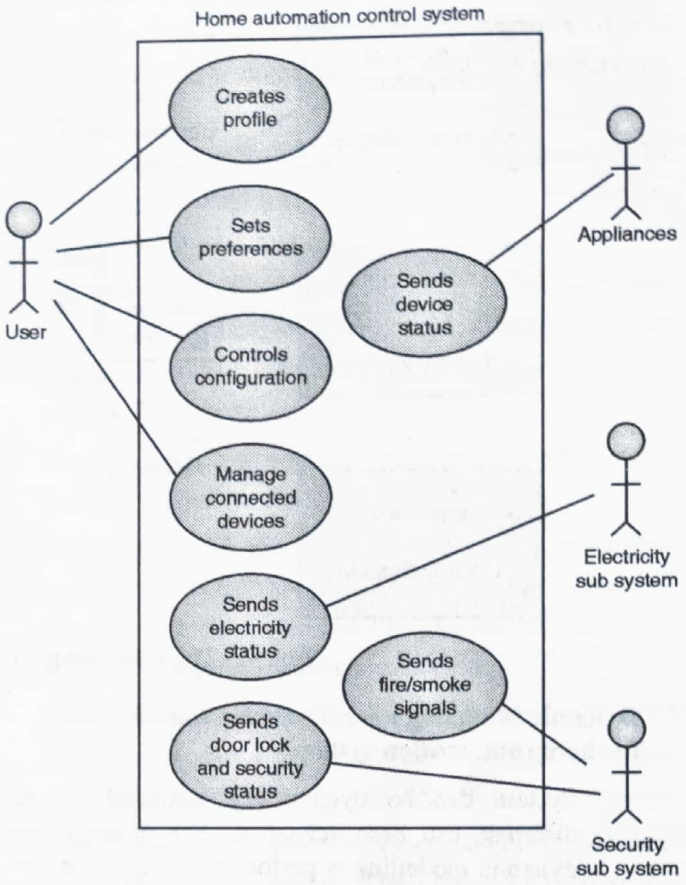


**Fig. 4.3.20 : Use Case: Home automation system**

☞ **Static Modeling : Class diagram for Home automation system**

- The static structuring of real time system can be depicted by a class diagram where primary subsystem can be shown as classes and their relationship can be shown as association between the classes.
- The class diagram for home automation system is shown in Fig. 4.3.21 where the automation control system is shown as a class stereotyped with software system.
- It requires co-operation from other subsystems like electricity subsystem, security subsystem and device subsystems.
- Inputs can be provided by a dedicated control panel sub system and a display subsystem used to get the output.

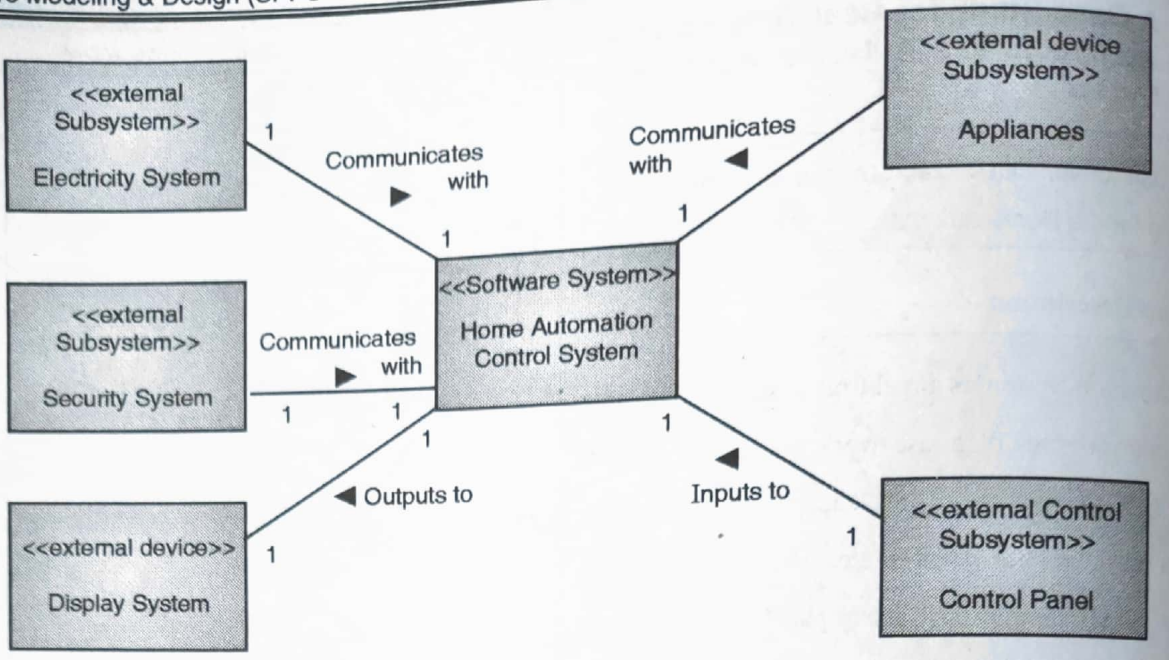


**Fig. 4.3.21 : Class diagram for home automation system**

☞ **Dynamic Modeling : A communication diagram for home automation system**

- System can be dynamically modelled for the different use case scenarios. For example one dynamic modelling is performed to show the how user interacts with the control system through control panel or mobile device.
- An abstract consolidated communication diagram is shown in Fig. 4.3.22.
- It shows the system instance while it is getting operated. The run time commutation between objects is depicted in Fig. 4.3.22.
- These objects are run time representation of home control system, control panel, security and electricity subsystems and display unit.
- Control panel sends user's selection and preferences to central home automation control object and receives back an acknowledgement.
- Detailed display and status information is sent to the display object which may be integrated with control panel object (composite object) but in communication diagram we are treating it as separate communication object.
- Electricity and security subsystems are represented as composite objects providing necessary status and even information to the central control system.
- Central home control system object sends instruction to other objects based on user's preferences and selection.

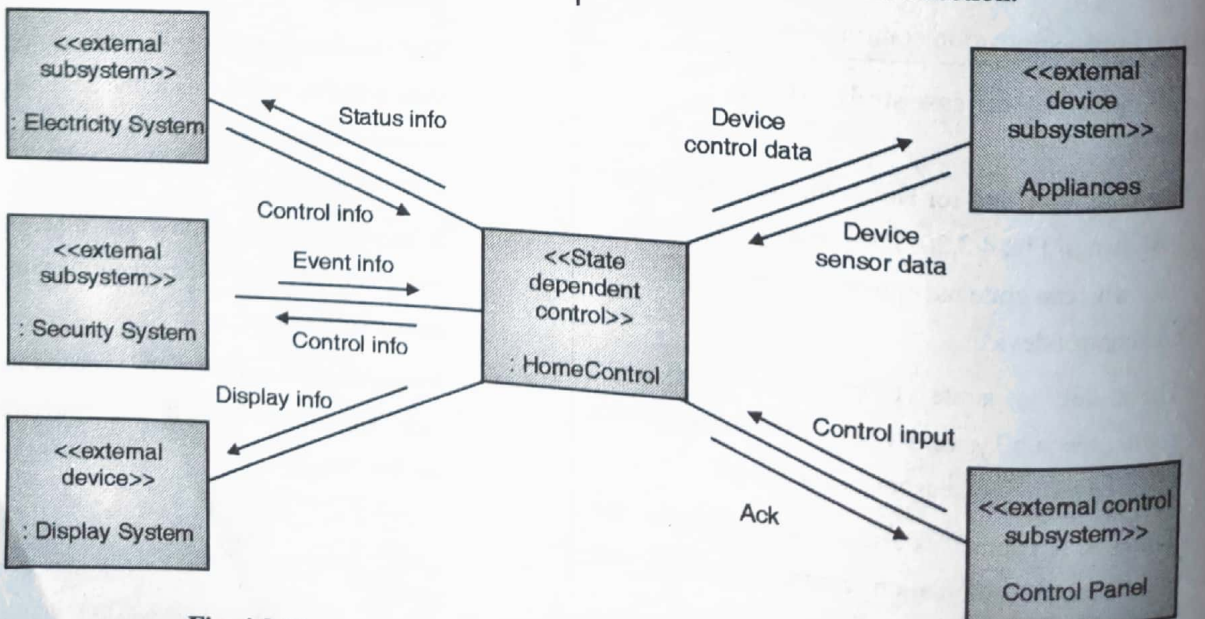


**Fig. 4.3.22 : Communication diagram for Home automation system**

## Summary

- **Software Architecture** or the architectural design of software is an overall architecture of the software system to be designed. This represents the entire system in the form of sub systems and their interconnection
- **Software architecture** depicts a modular design of the system in the form of components and their interconnection. Software architecture also refers the detailed internal design of each module and their interfaces.
- An **architecture design** for software system that focuses on components and their interconnection is called "Programming-in-the-large.
- An **architectural design** that focuses on internal design of each component is called "Programming-in-the-small.
- One of the major concepts associated with software architecture is **component based software architecture**. This is the structural perspective of software design.
- **Stereotypes** are unique kind of modelling elements specified by UML 2 and used to describe the nature of other modelling elements.
- **Stereotypes** specify the role characteristic of other modelling element. It is possible in an architectural design that one modelling elements plays multiple roles.
- Software architecture can be represented from different perspectives. These perspectives are called the different **views of software architecture**. These views use different modelling constructs and modelling elements to represent the design in pictorial form.
- Views are of software architecture can be classified as:
  a. Structural View
  b. Dynamic View or behavioural view
  c. Deployment View
- The **structural view** of software system is the static view and highlights the overall architecture of system.
- **Structural view** represents the overall architecture in the form of different organized subsystems connected with each other.
- **Structural view** can be represented by UML subsystem class diagrams and associations between them.
- The dynamic behaviour of system is captured by **dynamic view** of software architecture. This can be represented by a communication diagram.
- **Dynamic view** of system specifies in abstract that how the system will behave at run time. The emphasis is on communication between object and their message passing sequences.
- **Deployment view** captures system distribution in a networked environment.
- Today most of the system is web or enterprise applications and works as distributed systems. **Deployment view** specifies the location of each sub-system and their communication media.
- **Deployment view** can be depicted as deployment diagrams. It shows the deployment of subsystems at different network nodes and the type of network.
- **Deployment views** are more concerned about physical layout of software system. The idea is to capture the scope of system in terms of installation or deployment.
- A **design pattern** describe a recurring problem to be solved, a solution to the problem and the context in which that solution works.
- **Design patterns** are often termed as micro architecture and when the patterns are applied at higher level software design then called the **architectural pattern**.
- One of the simplest and widely applied architectural pattern is **client-server architectural pattern**. A client is the requestor of service and server is the provider of service.
- **Multiple client single service** is most common type of architectural pattern where multiple clients request for service to a single server.
- It is possible in client server architecture that along with multiple clients, there are multiple services or servers. An architectural pattern specifying such model is called **multiple client multiple service patterns**.
- **Multi-tier client-service architectural pattern** places one or more extra layer of service between conventional client and service.
- In **Multi-tier client-service architectural**, intermediate tiers take the services from upper layer and hence acts like a client for the upper layer services.

At the same time it can provide to services to lower level sub-systems and acts like a server.

- There are certain **communication patterns** also that can be applied in design along with architectural pattern. These patterns define a model of communication between different subsystems.
- Two important architectural communication patterns are :
  1. Synchronous message communication with Reply Pattern
  2. Asynchronous message communication with Call back Pattern
- In **Synchronous message communication with Reply Pattern** client requests for service to server and waits for reply. There is no message queue developed between that client and server and once server responds client proceeds with its operation.
- **Asynchronous message communication with call back pattern** is applied between a client and server communication where client can send a request to the server and continues with its normal operation
- Another architectural pattern is **Service Oriented Architecture** often termed is SOA. It is popular in modern days distributed web applications.
- A **service** itself can be defined as functionality which is very well-defined and self-contained. It does not depend on the state of other service or on the context of other services.
- **Service oriented architecture** is distributed software architecture and consists of multiple autonomous services.
- **SOA** is the extensions component based and client-server architectural pattern where one application components provides service to another component using the standard communication protocol over the network.
- The motivation behind **SOA** is that the services can be provided by different vendors and different services may use different technologies for its internal implementation.
- Design principles for SOA includes: Abstraction, Loose Coupling, Reusability, Service contract, Autonomy, Discoverability, Statelessness
- In Service Oriented Architecture, there is an intermediary between client and the server called **broker**.
- **Brokers** hide all the complexities related to obtaining the service from the client. So clients do not have to keep and maintain information about where and how to get the service. Broker helps client to locate and obtain the service.
- Broker is often termed as **Object Broker**. And this pattern of getting the service is called simply as ***Broker Pattern*** or as the ***Object Request Broker Pattern.***
- Two important functionalities provide by broker is : **Location transparency, Platform Transparency**
- **Location transparency** specifies that irrespective of the change in location, client should keep on getting the same service.
- **Platform transparency** specifies that clients do not need to know the hardware and software details of the services. Each service can be executed on a different hardware and software.
- **Real time systems** are special systems and has additional requirement during the design process. It has to deal multiple stream of input during its operations and has to quickly respond on real events.
- **Real time software architectures** are usually concurrent architectures and are state dependent. In concurrent software architecture, the system is organized or structured into concurrent tasks.
- **Real time software architectures** also defines interfaces between the concurrent tasks and the interconnection between the concurrent tasks.

## 4.4 Exam Pack (University and Review Questions)

☞ **Syllabus Topic : Introduction to Architectural Design**

**Q.** What is architectural design of a software system? *(Refer section 4.1)* (4 Marks)

☞ **Syllabus Topic : Overview of Software Architecture**

**Q.** Explain overview of architectural design. *(Refer section 4.1.1)* (2 Marks)

**Q.** Explain the importance of architectural design. *(Refer section 4.1.2)* (4 Marks)

**Q.** Describe component based software architecture in brief. *(Refer section 4.1.3)* (5 Marks)

☞ **Syllabus Topic : Object Oriented Software Architecture**

**Q.** What are architectural stereotypes ? *(Refer section 4.2.1)* (4 Marks)

**Q.** Describe the static structural view with class diagram. *(Refer section 4.2.2(B))* (5 Marks)

**Q.** With example, explain dynamic view of software architecture. *(Refer section 4.2.2(C))* (5 Marks)

**Q.** What is software architectural pattern? And why they are important in modern software development? Explain in brief. *(Refer section 4.3)* (6 Marks)

### ☞ Syllabus Topic : Client Server Architecture

**Q.** Explain client server architectural pattern.
*(Refer section 4.3.1)* (3 Marks) **(Feb. 2016 (In sem))**

**Q.** Explain multiple client single service architectural pattern with a suitable example.
*(Refer section 4.3.1(A))* (8 Marks)

**Q.** Explain multiple client multiple service architectural pattern with a suitable example.
*(Refer section 4.3.1(B))* (2 Marks)
**(Feb. 2016 (In sem))**

**Q.** Explain multi-tier client server architectural pattern. *(Refer section 4.3.1(C))* (4 Marks)

**Q.** What is important communication patterns applied in client-server architecture?.
*(Refer section 4.3.1(D))* (6 Marks)

**Q.** Explain synchronous communication pattern in client-server architecture with help of a diagram.
*(Refer section 4.3.1(D)(1))* (5 Marks) **(May 2016)**

**Q.** Explain asynchronous message communication with call back pattern with suitable example.
*(Refer section 4.3.1(D)(2))* (4 Marks)

### ☞ Syllabus Topic : Service Oriented Architecture

**Q.** What is service oriented architecture ?
*(Refer section 4.3.2)* (4 Marks)

**Q.** What are important design principles for service oriented architecture ?
*(Refer section 4.3.2(A))* (4 Marks)

**Q.** Explain location transparency and platform transparency in service oriented architecture.
*(Refer section 4.3.2(B))* (5 Marks)
**(Feb. 2016 (In sem))**

**Q.** Explain the Broker Patterns for design of service oriented architecture.
*(Refer section 4.3.2(B))* (5 Marks) **(Dec. 2016)**

**Q.** Write short note Web Service protocol.
*(Refer section 4.3.2(C))* (3 Marks)

**Q.** Write short note Web Service.
*(Refer section 4.3.2(C))* (5 Marks)
**(Feb. 2016 (In sem))**

**Q.** Write short note Web Service Registration Services.
*(Refer section 4.3.2(C))* (3 Marks)

**Q.** Write short note Brokering and discovery services.
*(Refer section 4.3.2(C))* (3 Marks)

**Q.** Explain Port, provided interface and required interface with an example. *(Refer section 4.3.2(D))* (5 Marks)
**(Feb. 2016 (In sem))**

### ☞ Syllabus Topic : Real Time Software Architecture

**Q.** Explain real time software architecture.
*(Refer section 4.3.4)* (5 Marks)

### ☞ Syllabus Topic : Real time Software Architecture

**Q.** Explain the important characteristics of real time software architecture.
*(Refer section 4.3.4(A))* (4 Marks)
**(Feb. 2016 (In sem))**

**Q.** Draw use case, class diagram and communication diagram for the Home automation system.
*(Refer section 4.3.4(B))* (10 Marks)

❑❑❑

UNIT V

CHAPTER 5

# Design Patterns

**Syllabus Topics**

Introduction to Creational design pattern – singleton, Factory, Structural design pattern- Proxy design pattern, Adapter design pattern, Behavioral – Iterator design pattern, Observer design pattern.

## 5.1 Design Patterns

### 5.1.1 Introduction to Design Patterns

**Q. What are design pattern and why they are important in modern software development. ? (4 Marks)**

- Design patterns are well defined problem solving strategies developed over the years of experiences in software designing, it provides a reusable solution to common problems which developers encounters in software development.
- This is a template that can be applied in design and can be transformed into coding while implementation.
- Design patterns enable software architects and developers to reuse their design and architectures.
- The primary motivation is: If we have spent significant resources in finding the solution of problem then why to spend the resource again for other similar kind of problems.
- We can exploit the wisdom and lessons learned by us or by other developers who have successfully solved the similar problem.
- With this philosophy of design reuse, design patterns largely increase the success rate of software projects. It minimizes the risk of failure in software development and enables the fast development of software system.
- Christopher Wolfgang Alexander, a well known architect describes the importance of design reuse as: "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of solution to that problem in such a way that you can use this solution a million times over, without ever doing it same way twice". He has experience of over 200 building project around world and made the statement related to his experiences of buildings and town architecture but in fact this statement equally applicable in software design. We will explore different design patterns in this chapter and will try to understand how there patterns serves as reusable solution to commonly occurring problems.

### 5.1.2 Elements of Design Pattern

**Q. Explain the essential elements of design pattern. (4 Marks)**

Each pattern in software designing has four essential elements. Based on these elements, design patterns are classified in Fig. C5.1.

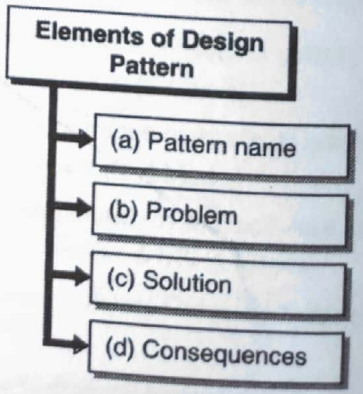


**Fig. C5.1 : Elements of design pattern**

**(a) Pattern name**

- The name acts as a handler to describe the design problem, solution and the consequences.
- The meaning of the name is logically related to the way it solves the problem.
- It is universally accepted among the developers and software architects and increases the design vocabulary.
- It helps developers to have an idea of the problem at a higher level of abstraction.
- A vocabulary filled with pattern name helps the developers to discuss the problem, share views, find solutions and perform documentation.

**(b) Problem**

- The context where a design pattern can be applied, is described by problem.
- It simply specifies, when to apply the pattern. It can also describe very specific design problem with details of how to represent algorithm as objects.
- It also specifies the list of condition that must be satisfied before applying the specific design pattern.

**(c) Solution**

- It specifies the design elements that form the design along with the predefined well specified strategy.
- The elements of design are also specified with their relationship, responsibilities and collaboration.
- It should be noted that solution does not describe application or problem specific solution but just a template that can be imposed on different situations.
- So a solution essentially describes how in general, different design element should be arranged to solve the similar types of problem.

**(d) Consequences**

- They are certain tradeoff that must be taken into consideration before applying the pattern. Over the years of experience along with the design solution, certain repeated tradeoffs have also been found.
- These are called consequence and help us to decide on the result of applying the pattern. Consequences help in understanding the cost and benefits of applying specific pattern.

## 5.1.3 Classification of Design Patterns

**Q. What are the major classification of design pattern? Explain in brief with few example patterns under each class. (6 Marks)**

Designs patterns are categorized in three groups are shown in Fig. C5.2.

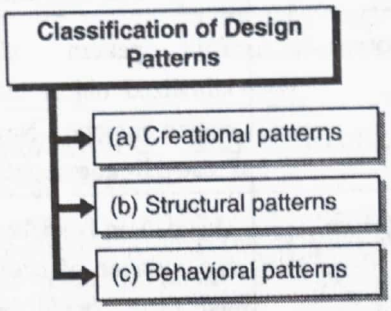


**Fig. C5.2 : Classification of Design Patterns**

**(a) Creational patterns**

- The primary concern of creation design pattern is to abstract the instantiation process.
- When it is desired that system should be in dependent complexities of object creation and representation, creational design patterns are applied.
- We will discuss creational design patterns in greater detail in next section.
- For now, we will briefly mention the popular patterns under creational design pattern. Table 5.1.1 show is the list of popular creational design patterns.

**Table 5.1.1**

| Sr. No. | Design Pattern name | Description |
|---|---|---|
| 1. | Abstract factory | This pattern provides an interface for creating of object which are related or dependent. It does not specify their concrete classes. |
| 2. | Builder | The pattern separates the object representation from the complexity object creation. It allows construction process to create various representations. |

| Sr. No. | Design Pattern name | Description |
|---|---|---|
| 3. | Factory method | It specifies an interface for the creation of single object but allows subclasses to decide on which class to instantiate. In other words it creates instance of several derived classes. |
| 4. | Prototype | This pattern allows fully initialized object instance to be copied or cloned. New objects are created by copying the prototype. |
| 5. | Singleton | This pattern is applied when only one instance of class is needed, and there should be only one global access point to it. |

➜ **(b) Structural patterns**

- Structural patterns are applied when the primary concern of design is to compose large structures from classes and objects.
- In other words this pattern concerns with class and object composition. Inheritance is used by structural class pattern to compose interfaces.
- We will explore different patterns with example in next section to clearly understand structural pattern. The list of different pattern belonging to structural pattern is given in Table 5.1.2.

**Table 5.1.2**

| Sr. No. | Design Pattern name | Description |
|---|---|---|
| 1. | Adapter or Wrapper | It matches interfaces of different classes. Interface of a class is converted into another interface as client expects. Using the adapter two or more classes can be used together that could not otherwise because they have incompatible interfaces. |
| 2. | Bridge | Separates or decouple the abstraction from its implementation in such a way that they both can vary independently. |
| 3. | Composite | Part whole hierarchies are formed to represents objects in a tree like structure composite enables the client to treat single object and composition of objects uniformly. |
| 4. | Decorator | Additional responsibilities are attached to an object dynamically without changing the interface. This serves as an alternative to sub classing for extending functionalities |
| 5. | Facade | A sub system can have many interfaces. Facade enables to have single interface provided as unified interface to set of interfaces. This higher level of interface makes it easier to use the subsystem facade often considered as pattern applied when there is need of single class representing entire sub system. |
| 6. | Fly weight | Large number of similar object can be shared efficiently. |
| 7. | Proxy | A place holder for another object that controls the access to main object. |

➜ **(c) Behavioral patterns**

- The main concern of behavioral pattern is to assign algorithms and responsibilities to objects.
- A behavioural pattern is a pattern to describe class, objet and their communication.
- It defines the control flow at runtime between the objects and makes it easier to understand how the objects are interconnected.
- Behavioural pattern describes the communication between objects.
- The list of important patterns under the behavioural pattern is provided in Table 5.1.3.

Table 5.1.3

| Sr. No. | Pattern name | Description |
|---|---|---|
| 1. | Chain of responsibility | Specifies the way of passing request between chains of objects. It chains the objects in such a way that if one object does not handle the request, it is passed to another object. |
| 2. | Command | Command enables the request to be encapsulated as an object. |
| 3. | Interpreter | Interpreter allows to include language element in design so that to interpret complexities. |
| 4. | Iterator | Iterator allows the access to the element of an aggregate object. Access is sequential and does not reveal internal representation of object. |
| 5. | Mediator | It encapsulates the coordination logic in an object that defines how a group of objects interact with each other. The objective of mediator is to implement loose coupling. |
| 6. | Memento | Objects internal state is capture and externalized with breaking the encapsulation. Once externalized it also allows to restore the previous state of object. |
| 7. | Observer or publisher/subscriber | Allows one to many dependence between objects. If the state of one object is changed, all the dependent object (subscriber) are notified about the event. |
| 8. | State | It allows object to change its behavior whenever the internal state of the object changes. |
| 9. | Strategy | A number of algorithms are defined, encapsulated and made interchangeable. Algorithms are allowed to vary independently irrespective f client's use. |
| 10. | Template method | The template or skeleton of an algorithm is defined in a class, then subclasses are allowed to vary few steps without changing the original structure. |
| 11. | Visitor | Visitor allows new operations to be defined but does not change the classes of the element on which it operates. |

## 5.1.4 Documenting and Describing the Design Pattern

**Q. What are the major elements of documenting and describing the design pattern? (5 Marks)**

- There are so many design patterns are existing and each pattern can be applied in different scenario.
- The clearly understand the application of design pattern, it is important to describe the pattern in very formal and well defined way.
- A consistent format is followed to describe the patterns among developers, Learners and students.
- Patterns are expressed into well specified structure that makes it easier to share, understand and express,

☞ **Primary section**

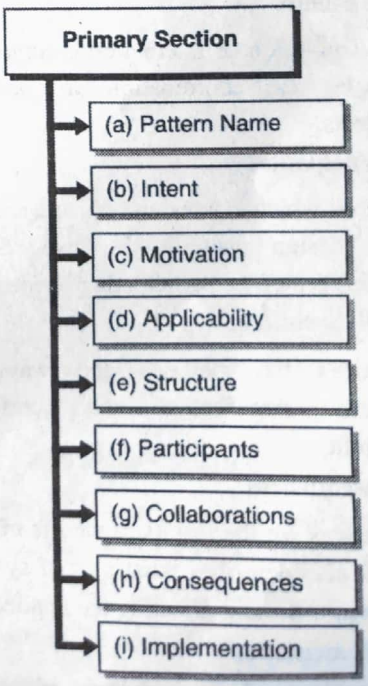


**Fig. C5.3 : Primary section**

➜ **(a) Pattern Name**

Pattern name is very logical and related with the application and strategy of the pattern. It belongs to the design vocabulary and are that's why is considered important.

➜ **(b) Intent**

This is brief description of pattern. It exactly specifies in short that what the design pattern does and what problem it solves.

➜ **(c) Motivation**

- It exactly describes the scenario where a problem is solved by applying pattern on structure of classes and objects.
- Scenario is also considered as an abstract description of the pattern and they are detailed in subsequent sections.

➜ **(d) Applicability**

- Applicability is description of more practical situation where a design pattern can be applied.
- It may also include example of other successful application of same pattern and example of some poor design in the same situation additionally it describes how the situations are identified.

➜ **(e) Structure**

- Notations of Object Modeling Techniques (OMT) are used for pictorial representation of classes in the pattern.
- Collaboration or interaction diagrams are used to display the coordination and arrangement of objects.

➜ **(f) Participants**

It describes which classes and objects are participating in the design pattern. It also describes the responsibilities of the participating elements.

➜ **(g) Collaborations**

It describes the methods and ways by which participants contributes and performs their responsibility.

➜ **(h) Consequences**

Consequences are the results, tradeoffs of applying the pattern. It also describes what part of system you can vary in design without affecting the applied pattern.

➜ **(i) Implementation**

Implementation include guidelines, pitfalls, hints and techniques of implementing the pattern.

## Syllabus Topic : Introduction to Creational Design Pattern

## 5.2 Creational Pattern

**Q. What is creational pattern? (2 Marks)**

- Creational design patterns are used to provide an abstraction of instantiation process.
- Creational pattern separates the system from the complexities of object creation, object instantiation and its representation.
- Creational pattern plays important role when system evolves to use composition more often than inheritance
- Creational patterns allows the flexibility in what is getting created, who is creating, how it is getting created and when it is getting created.
- Creational patterns allows configuration of object as product where the configuration can be specified statically at compile time or dynamically at run time.

☞ **Two important patterns under creational pattern**

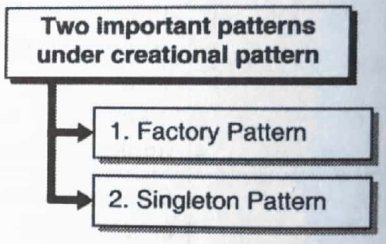


**Fig. C5.4 : Important patterns under creational pattern**

## Syllabus Topic : Factory

### ➜ 5.2.1 Factory Pattern

➜ **(Feb. 2016)**

**Q. Write short note Factory Pattern.**
**SPPU - Feb. 2016(In sem), 5 Marks**

**Q. Explain factory pattern. Describe its intent, motivation and implementation with suitable example. (5 Marks)**

- Factory pattern provides an interface for creation of object which are related or dependent. It does not specify their concrete classes.

☞ **Intent of Factory Pattern**

(a) Creating object without revealing instantiation logic to the client.

(b) Provides a way to refer newly created object through a common interface.

### Motivation of Factory Pattern

- Factory is the popular and most used design pattern in modern application development.
- Languages like java, c ++, c# can be used to implement factory pattern.
- There are variations in factory pattern. One variation is called Factory method and another is abstract factory.

### Implementation of Factory Pattern

- Implementation of factory pattern is quite simple. Client requires a product but it does not create the product directly using the new operator which is used in languages like java to create the object. Client instead asks a dedicated factory object for a new product. Client specifies which type of object it needs to the factory.
- The factory creates or instantiates a new product and provides the newly created concrete product to the client.
- The newly created concrete product is casted from abstract product class or interface.
- Client can use the product as an abstract product. Client does not need to know about its concrete implementation.
- The client, factory, product abstract, product (Product interface) and their relationship is shown in Fig. 5.2.1.

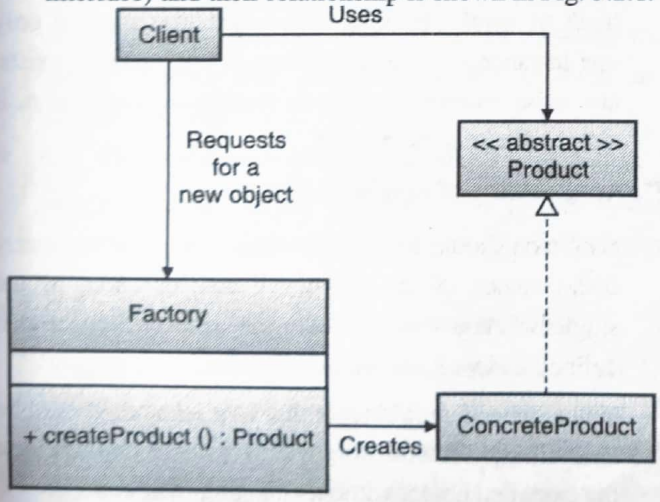


**Fig. 5.2.1 : Implementation of Factory Pattern**

- Fig. 5.2.1 shows the implementation of Factory Pattern. The class diagram shows a Factory class offering a public method create Product () that returns the abstract Product reference that client can use.

### Applicability of Factory Pattern

- Taking an example of drawing framework application, applicability can be understood for the Factory Pattern.
- A drawing application deals with shapes and drawing elements. We can consider drawing framework as client and shapes as products.
- Different shapes can be derived from an abstract Shape class. There is declaration of operations like draw (), colour (), fill () and move () but abstract class does not provide implementation of these operations. The concrete Shape class must implement these operations.
- Take an example where user interact with the drawing framework and issues a command to draw a new Square Shape. The parameter is received by framework as string "Square".
- Then drawing framework acts as client and requests factory to create a new concrete shape by sending the parameter 'Square" then factory creates a new square and returns it to drawing framework, which is casted as (derived from) abstract Shape class.
- The client which is drawing framework is actually not aware of the concrete type Square but treats the newly created object as Shape.
- So according to pattern specification client uses the object as casted to the abstract class (or interface) but is not aware of concrete object type.
- The scenario is depicted in Fig. 5.2.2 client is drawing framework and product is shape.

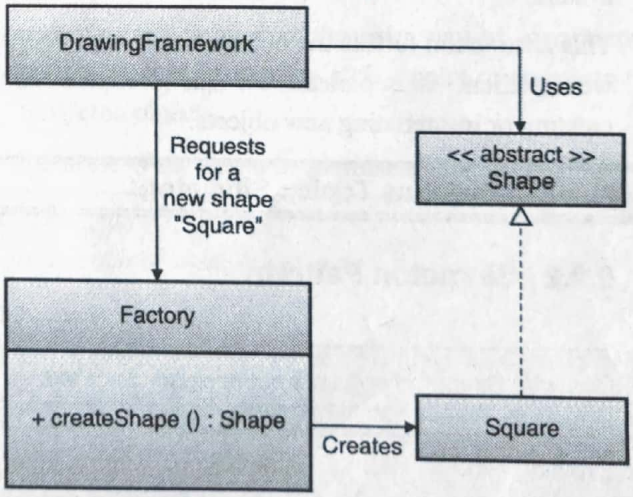


**Fig. 5.2.2 : Applicability of Factory Pattern**

### Conclusion of Factory Pattern

Framework will remain unchanged and new shapes can be added. For example adding a new shape Rectangle will not affect the client "Drawing Framework".

### Sample Code of Factory Pattern

- A sample code for Factory is provided here. We name it as "ShapeFactory" that has one public method named as create Shape ().

```
public class ShapeFactory {
   public Shape createShape (String shape) {
      if (shape.equals ("Circle"))
         return new Circle ( );
      if (shape.equals ("Square"))
         return new Square ( );
      if (shape.equals ("Rectangle"))
         return new Rectangle ( );
      ...// so on for other shapes
   return null; // in – case of unexpected string
   }
   .....
}
```

- The method createShape ( ) returns an object of type Shape.
- The method createShape ( ) expects a string argument based on which the new shape will be created.
- It should be noted that client does not use the new operator to create the Shape, instead it directly calls createShape () method of Factory class.
- When a new shape needs to be added, client does not modifies its code. Only Factory is updated for creating the new shape and a new shape class is added in the system.
- This abstraction fulfills the objective of Factory pattern where client is separated for the complexities of creating or instantiating new objects.

**Syllabus Topic : Singleton**

## 5.2.2 Singleton Pattern

→ (Feb. 2016)

**Q.** Describe Singleton pattern and its significance with an example. **SPPU - Feb. 2016 (In sem), 5 Marks**

**Q.** Explain the intent, motivation, applicability, implementation and consequence of singleton pattern. **(6 Marks)**

**Q.** What is singleton pattern ? Explain one example scenario where you will singleton pattern to get applied. **(6 Marks)**

- Singleton pattern is extremely useful in modern application scenarios. It ensure creation of only one intended object.
- Singleton pattern is applied when only one instance of class is needed, and there should be only one global access point to it.

### Intent of Singleton Pattern

Singleton ensures that a class should have only one instance, and provides a global access point to it.

### Motivation of Singleton Pattern

- Sometimes it is required that a class should have only one instance. For example there should be exactly one Window Manager in a GUI application, or there should be only one instance of Virus Scan running in the system.
- Singleton pattern is one of the simplest and interesting design pattern. It is exciting to observe that how it is ensured during the coding that a class should have only are object at a time.
- In singleton pattern, a class is responsible for instantiating itself and class is also responsible for making it sure that only one instance should be created. It also ensures that there should be a global access point to that instance.
- The only one instance of the class is used from everywhere in the application, and other classes cannot invoke the singleton class constructor directly.
- Singleton allows a class to monitor itself. A class keeps track of itself, checking every time that there is only one instance of class is existing. If one instance exists, and class receives request to instantiate itself again, it simply discards the request.

### Applicability of Singleton Pattern

- Singleton should be applied when there must be exactly one instance of the class. A user or client of the singleton class should access the instance from a well defined and well known access point.
- When there is requirement that sole instance should be extensible by sub classing and user of the class can use the extended object without changing their code.

### Participants of Singleton Pattern

- Singleton defines a special method or operation in class that allows its client to access the unique instance.
- A class itself is responsible for creating its own unique instance. So there is no other participant when it comes to creating new instance of the class.

☞ **Collaboration of Singleton Pattern**

There is no collaboration of classes to access the Singleton instance. Client access the Singleton instance only through Singleton's Instance operation.

☞ **Consequences/Advantages of Singleton Pattern**

There are many advantage of applying singleton pattern in specific scenarios.

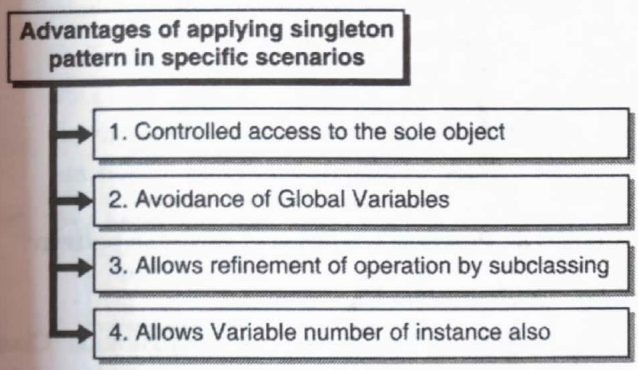


**Fig. C5.5 : Advantages of singleton pattern**

➜ **1. Controlled access to the sole object**

Singleton class is supposed to encapsulate its sole object or sole instance. It provides a way to have very strict access to the sole instance. This controlled access is provided by a well formulated operation defined in class itself.

➜ **2. Avoidance of Global Variables**

Singleton pattern make the system free from global variables. It reduces the name space by avoiding the global variable that otherwise needed to keep the reference of sole object.

➜ **3. Allows refinement of operation by subclassing**

Singleton class can be extended and operation are refined. Application can easily be configured for instance of extended subclass.

➜ **4. Allows Variable number of instance also**

Singleton not always used to create only single instance. Logic can be written to limit the number of instances to a variable number that can even be varied at run time. For example, if an application requires that only five instances should exist for a class, Singleton allows it to happen.

## 5.2.2(A) Implementation of Singleton Pattern

- Different programming language may use different ways for supporting and implementing Singleton pattern.
- In general, implementation involves a static member that can store the reference of Singleton object, a public method that returns a reference to the static member and most importantly a private constructor that actually creates the singleton sole object and can be accessed only from inside the singleton class.
- A UML class diagram for singleton class is shown in Fig. 5.2.3.

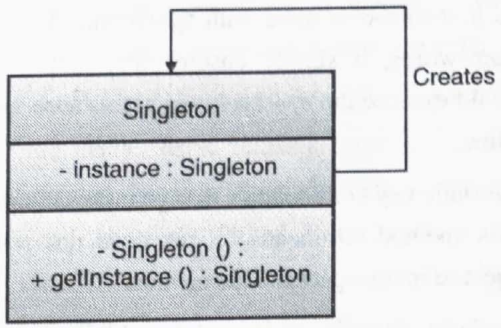


**Fig. 5.2.3 : A UML class diagram**

- getInstance() is the single access point for Singleton's client and is responsible for creating unique instance of its class.
- Method getInstance() encapsulates the necessary logic for ensuring that only one instance of the class should be created.
- getInstace() operation is the only way of instantiating the class hence provides a global point of access to the Singleton object.

☞ **Example of singleton implementation**

```
class Singleton {
   private static Singleton instance;
   private Singleton () {
   ----------
      } // Singleton Constructor
      public static synchronized Singleton getInstance()
      {
         if (instance = = null){
            instance = new Singleton();
            // private constructor is called
         }
      return instance;
}
      public void more operation ()
      {
         ----------
      }
}
```

- In above code sample, it can be observed that operation getInstance encapsulates the logic of creating sole object. Code inside getInstance ensures that only one instance of Singleton class, should exist.
- getInstance method is defined as synchronized for a purpose. It ensures that when one thread is executing the synchronized method of the class, all other threads if invoke synchronized method has to be blocked until the first thread is done with synchronized method. In other words, it simply ensures that only one client should execute the synchronized getInstance method at a time.
- The static keyword denotes that getInstance method is a class method. It means, client does not require an object to invoke getInstance method.
- Similarly there is static variable declared, named as instance. The static keyword here denotes that the variable instance is a class variable and a single global copy of this variable is maintained.
- Consider one example where a client code calls the public method getInstance, as soon as application starts. Since getInstance method is getting called for the first time, the value of variable instance is null.
- The 'if' condition inside getInstance method becomes true and Singleton object is created by calling a private constructor. Client gets the reference of that sole object.
- For all subsequent client's request (a call to getInstance() method), 'if' condition becomes false since the static variable instance now stores the reference of previously created sole object. This code logic ensures that single instance should exist throughout the application.
- Notice that the constructor is declared as private, so that external client's cannot access the constructor directly, and the single point of access is global method getInstance so if client needs some service say, moreOperation() from Singleton class, it should access in the following manner.

```
Singleton.getInstance().moreOperation()
```

Here Singleton.getInstance() will return a sole object and then moreOperation() can be invoked.

☞ **Applicability of singleton pattern**

Singleton pattern is applied when there is requirement that there must be exactly one instance of a class and it can be accessed by a single global access point. Some practical scenarios are listed in Fig. C5.6.

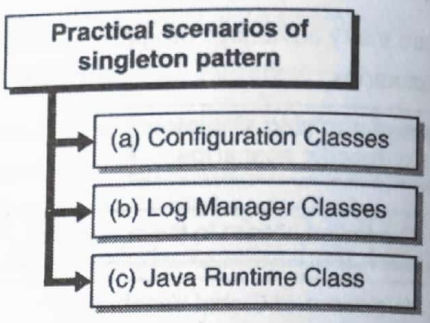


**Fig. C5.6 : Practical scenarios of singleton pattern**

➜ **(a) Configuration Classes**

- Certain application require Configuration Class that is responsible for configuration setting of the application.
- A natural choice for implementing configuration class is Singleton pattern so that instance of configuration class can be accessed globally.
- Also the instance can be kept as cache object, and all the configuration parameters are uniquely saved or cached in single object throughout application.

➜ **(b) Log Manager Classes**

- Singleton patterns are used to implement Log Manager Class that provides global logging access point to different components of the application.
- There should be single instance of Log manager or Logger Class in the application that enables only one component of the system to print its log at a time.

➜ **(c) Java Runtime Class**

- JVM associates exactly one instance of Runtime class with each running Java application.
- Java application can obtain the instance by calling Runtime.getRuntime(). (Refer Runtime class in Java documentation)

**Syllabus Topic : Structural Design Pattern**

## 5.3 Structural Patterns

**Q. What is structural pattern?** **(2 Marks)**

- Structural pattern deals with formation of large structures with the help of smaller classes and objects.

- Interface or implementations are composed by using inheritance. For example multiple inheritance can be used to mix two or more classes into one.
- The composed class has the mixed properties formed from the properties of parent class.
- The pattern is often used when different developers develop individual classes and later on the classes are combined to compose more useful classes of the system.
- Structural pattern describes the ways to combine existing object to realize new object with more useful functionalities.

☞ **Two important structural pattern**

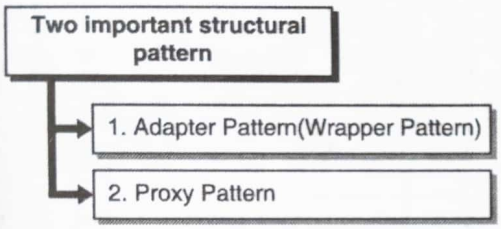


**Fig. C5.7 : Structural pattern**

## Syllabus Topic : Adapter Design Pattern

### ➜ 5.3.1 Adapter Pattern (Wrapper Pattern)

**Q.** Explain the intent, motivation, applicability, implementation and consequence of Adapter Pattern. **(6 Marks)**

- The dictionary meaning of adapter is "Device that enables something to be used in a way different from that for which it was intended" or "which makes different pieces of apparatus compatible". This pattern encourages the use of such adapter class that provides adapting between classes and objects.
- Adapter class works like adapter in real world and serves as an interface or bridge between two objects.

☞ **Intent of Adapter Pattern**

- Converts interface of a class into another interface that client expects.
- Adapter provides a way for incompatible class to work together.
- Wrap an existing class with new Interface.

☞ **Motivation of Adapter Pattern**

- To understand the motivation, take the example of drawing framework that we discussed earlier in factory pattern.
- Drawing framework allows to draw shapes using graphical elements and form picture and diagrams.
- Drawing framework uses a key abstraction called graphical object, that is nothing but an editable shape and has the capability of drawing itself.
- The shape is an abstract class server as an interface for graphical object. Drawing framework design subclasses of shape for different kinds of graphical object like circle shape, Rectangle shape, Square shape etc.
- Now note that circle shape, rectangle shape, line shape etc are very basic classes and can easily be implemented because their drawing capabilities are same and inherently limited. But a complex class like text shape should support complex operations like draw and edit texts.
- Now suppose there is another class already implemented named Text View for different purpose. It has been realized the Text View has operation like **display Text** and **edit Text.**
- Developer's would like to use TextView to implement TextShape but Text View might belong to another module and has not been designed with shape class in mind. So the TextView and shape classes are incompatible and their objects cannot be used interchangeably.
- Now the requirement is to change the TextView class such that it conforms to shape Interface.
- But how can we change TextView class that belongs to other module on which we do not have control or we do not have source code.
- One solution is change or redefine TextShape so that it conforms or adapts TextView interface to shape's. There are two ways of achieving this objective
  (1) Inherit shape's interface and TextView is implementation.
  (2) Implement Text shape in terms of TextView's interface and compose TextView object within TextShape.
- Here, Text shape becomes an adapter, the scenario is depicted in Fig. 5.3.1.

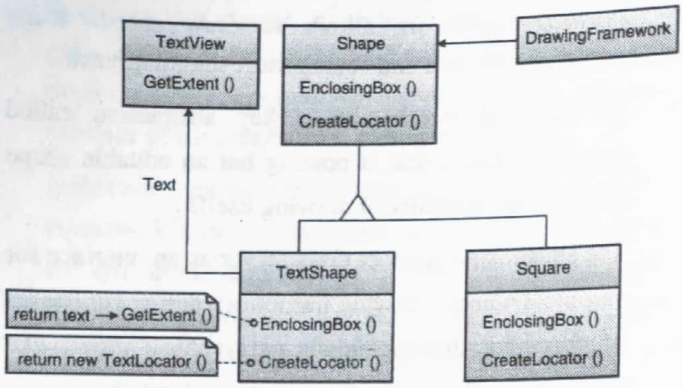


**Fig. 5.3.1 : Adapter pattern in drawing framework application**

- Fig. 5.3.1 illustration how Enclosing Box operation declared in class shape are converted to get Excellent operation defined in TextView. In this way, TextShape adapts TextView to the shape interface and drawing framework can reuse the incompatible TextView class.
- As per adapter pattern's specification, adapter is responsible for provided a functionality that adapted class does not provide. In our example, a user can relocate or drag the shape.
- But TextView does not have functionality. So TextShape can includes this missing functionality by implementing shape's create locator operation. This operation returns an object of appropriate locator subclass.
- The above scenario can be explained in more simplified way in Fig. 5.3.2.

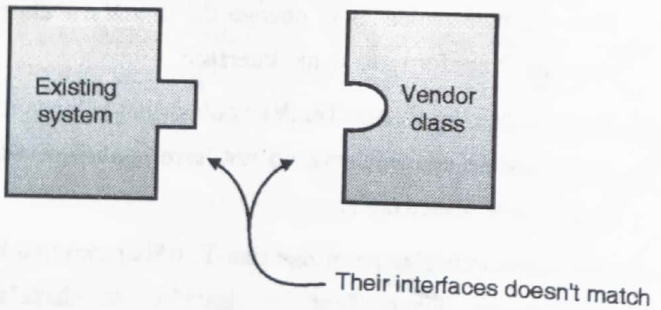


**Fig. 5.3.2**

- Suppose left side is your existing software system and right side is new vendor. Class library that has incompatible interface.
- You don't wish to change the code of your existing system and even don't have control over vendor's class source code.
- So you have written an adapter class that adapts new vendor interface to something that you expects.

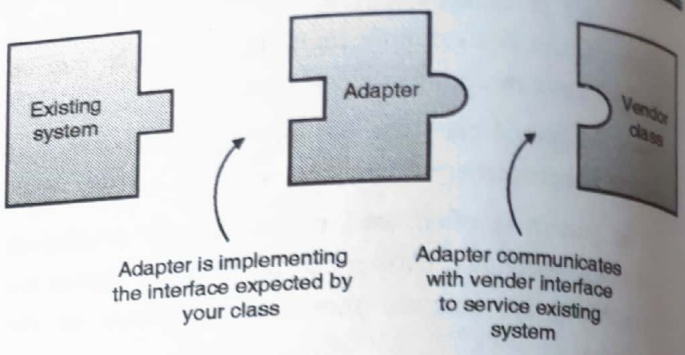


**Fig. 5.3.3**

- Finally your system works with the help of adapter and gets service from vendor class.

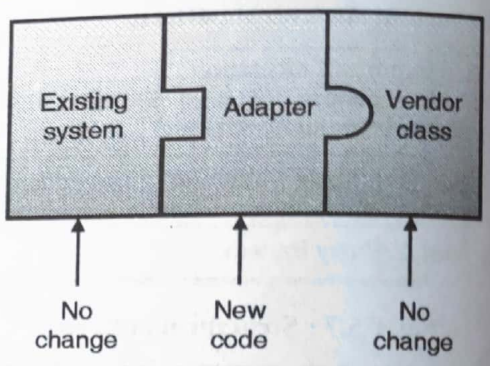


**Fig. 5.3.4**

### ☞ Applicability of Adapter Pattern

- Adapter pattern is applied when we want to use an existing class but its interface becomes incompatible with the new class we need.
- Adapter pattern is applied when we want to create reusable class and at the same time we want to enable the reusable class to communicate with other unrelated classes.
- Adapter pattern is applied when many existing subclasses can be used in the system, but it is not practical to adapt their interface by sub classing every one.

### ☞ Structure of Adapter Pattern

- The structure of adaptor is derived from its standard definitions : "The adapter pattern converts the interface of an class into another interface that client expects. Adapter allows class to work together that couldn't otherwise because of incompatible interfaces."
- This definition allows us to draw a structural class diagram for the adapter pattern, as shown in Fig. 5.3.5.

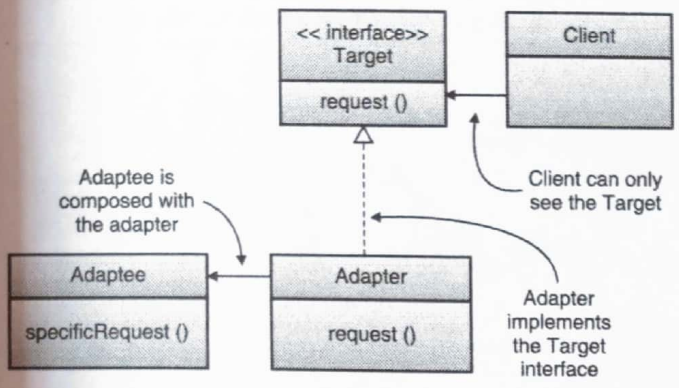


**Fig. 5.3.5 : Class diagram for adapter pattern**

☞ **Participants of Adapter Pattern**

Four participants are listed in Fig. C5.8. They are specified with the name of class and interface from previous example.

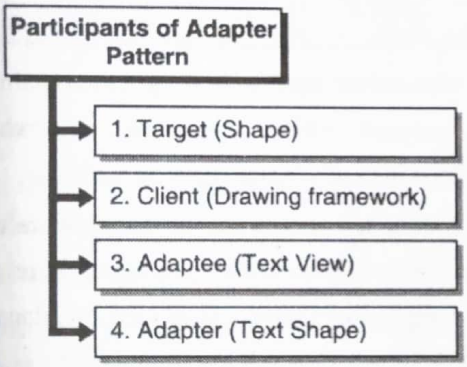


**Fig. C5.8 : Participants of Adapter Pattern**

➜ **1. Target (Shape)**

Application specific interface that client in already using.

➜ **2. Client (Drawing framework)**

The existing system with objects conforming to the target interface.

➜ **3. Adaptee (Text View)**

It specifies an existing interface that needs adapting.

➜ **4. Adapter (Text Shape)**

It adapts the interface of adaptee to the target interface.

☞ **Collaboration of Adapter Pattern**

The existing system or client issues requests on an adapter instance. Then adapter calls adaptee to serve the requests.

☞ **Consequences of Adapter Pattern**

- Adapter overrides few behavior of adaptee, since adapter is a sub class of a adaptee.
- No addition redirections of references is needed to get the adaptee and only one new object gets introduced.

### 5.3.1(A) Implementation of Adapter Pattern

- We will take an example of set of hypothetical classes to cover implementation aspect of adapter.
- We will define a Leopard interface that declares two operations : hunt ( ) and climb( ).

```
public interface Leopard {
public void climb ( ) ;
public void hunt ( ) ;
}
/ * so all the implementation of Leopard
can climb and hunt * /
```

- Now, we will declare subclass Leopard, the snow Leopard

```
public class SnowLeopard implements Leopard {
  public void climb ( ) {
    System·out·println ("climb");
  }
  public void hunt ( ) {
    System.out.printin ("Hunt");
  }
/* for the sake of simplicity, these two operation just prints
what it is doing */
```

- Now let's consider the new classes that needs to be cooperated in the system.

```
public interface Tiger {
  public void swim ( );
  public void hunt ( );
}
/* tigers don't climb, they swim*/
/* tigers hunt but less efficiently */
```

- So all the implementation of tiger can swim and hunt.

```
public class SiberianTiger implements Tiger {
  public void swim ( ) {
    System.out.println ("I am swimming") ;
  }
  public void hunt ( ) {
    System.out.println ("hunting large animals only");
  }
/* Like Leopard, the concrete implementation of Tiger, just
prints out it's action*/
```

- Now let us assume we have Leopard objects in our system.
- For some reason we are short on Leopard objects and wants to use Tiger objects in their place. But we cannot use Tigers directly as they have different interface. So we need a Tiger adapter class. Let us write the adapter.

```
public class TigerAdapter implements Leopard {
   /* First you need to implements the interface of the type
you are adapting to.
   This is the interface that client want to see */
   Tiger tiger ;
   public TigerAdapter (Tiger tiger) {
      this.tiger = tiger ;
   }
/* In above code, we got a reference to the object that we are
adapting, so we have used constructor for this purpose */
   public void climb ( ){
      tiger.swim ( );
   }

/* In above code, the climb translation is provided by simply
calling swim ( ) method. This is required because we need to
implement all the methods in the interface a class is
implementing *,
   public void hunt ( )
   {
      for (int i = 0; i < s ; i + +)
      {
         tiger.hunt ();
      }
   }
/* both the interfaces have hunt method
   but their internal implementation may differ */
A tiger may use different hunting strategy */
}
```

- This example of hypothetical system containing Tiger and Leopard objects provides somewhat more simplified idea of adapter implementation.
- On a final note, we will revisit the structure of adapter pattern and will draw the class diagram in relation with this example. Let's have a final look on adapter pattern in Fig. 5.3.6 which is self explanatory.

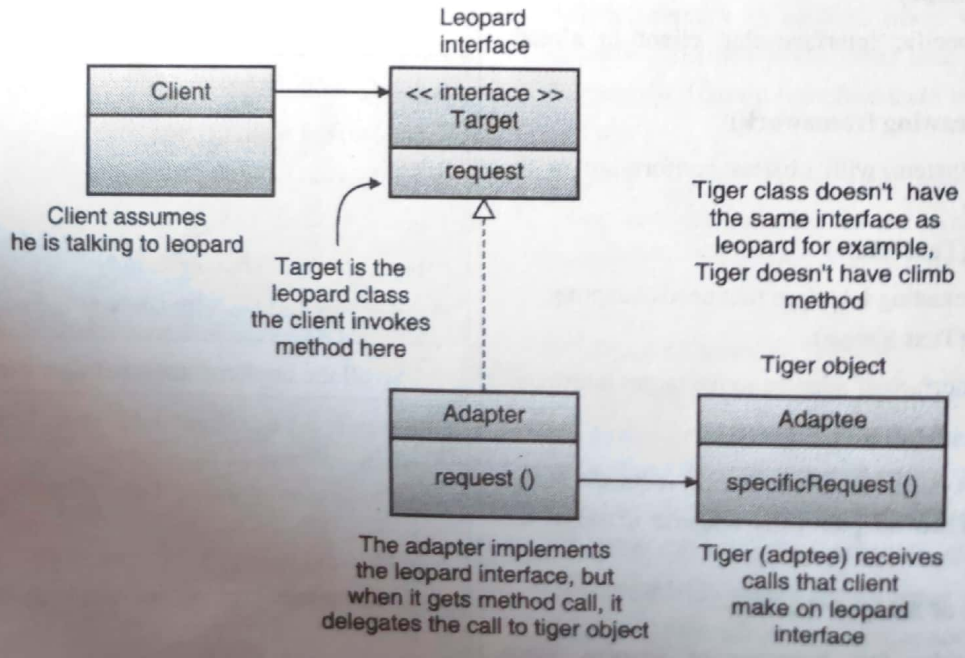


**Fig. 5.3.6 : Class diagram illustration with example**

**Syllabus Topic : Proxy Design Pattern**

## 5.3.2 Proxy Pattern

→ (Feb. 2016)

**Q.** Write short note proxy pattern.

SPPU - Feb. 2016(In sem), 5 Marks

**Q.** Explain the intent, motivation, applicability, implementation and consequences of proxy pattern. **(8 Marks)**

- In general proxy is a class functioning as an interface to something else. Proxy can provide an interface to things like : a file, a network connection or any other expensive resource which is hard to duplicate.
- Proxy can be considered as an agent or wrapper that communication to the client and provided an abstraction of real serving object behind the scenes.

### Intent of Proxy Pattern

The intent of this pattern is to provide a surrogate or placeholder for another object to control access to it.

### Motivation of Proxy Pattern

- Sometimes there is need of controlled access to an object. The reason of controlling access to an object in to temporarily delaying or deferring the full cost of creation and initialization.
- In some scenarios, it is required that costly objects should not get initialized entirely. Costly objects involves objects of greater importance and which requires significant amount of resources for its full operation.
- For the time being only few methods are required from costly objects, so the entire object should not get initialized, only few methods can be exposed by a proxy light weight object. This proxy light weight objects implements the same interface as heavy or costly objects.
- To understand the motivation, take the example of gallery application that shows the images available under different sections. When someone opens gallery, program must be able to list image belonging to different folder, but all the image do not need to be loaded at first.
- Once the user opens the sections and the image, then only high resolution image is rendered by gallery application. Once loaded fully, more operations can be performed on that image. So as soon as gallery is opened, the proxy object responds by providing limited information about the image.
- A high resolution image can be loaded and rendered at later stage, and then (if required) actual object (heavy object) starts responding. Fig. 5.3.7 depicts this scenario.

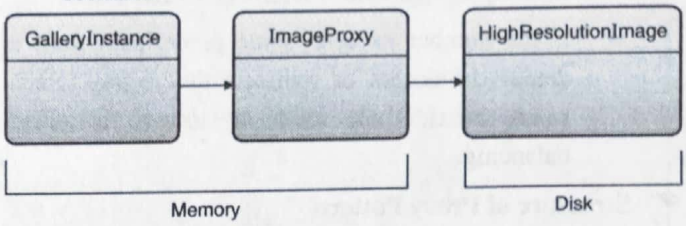


**Fig. 5.3.7: Simplified illustration of proxy pattern on gallery application**

### Applicability of Proxy Pattern

Proxy pattern, as described earlier is applicable when there is need of a more versatile and efficient reference to an object than a simple pointer. A list of common situations is provided in Fig. C5.9 for the illustration of proxy pattern applicability.

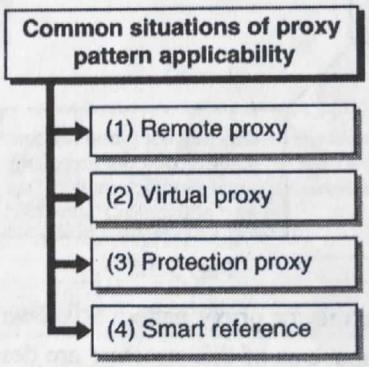


**Fig. C5.9 : Common situations of proxy pattern applicability**

#### → (1) Remote proxy

- Remote proxy object provides the local representation of an abject that actually belongs to a different address space.
- One popular example is Java RMI stub objects. So client invokes method on stub and stub on behalf of client invokes method on remote object.

#### → (2) Virtual proxy

- Virtual proxy delays the creation of costly and heavy objects and the actual objects are created by more specific request on demand.
- The gallery application discussed in earlier is a example of virtual proxy.

➜ **(3) Protection proxy**

Protection proxy object control access to original object. For example operating systems kernel is sometimes implemented as kernel proxies that provides access control to critical kernel operations.

➜ **(4) Smart reference**

- It provides more sophisticated operations like tracking the number of reference to an object.
- If the number exceeds some predefined limit, a denies the access or redirects the request. Such smart proxies are used in servers for load balancing.

☞ **Structure of Proxy Pattern**

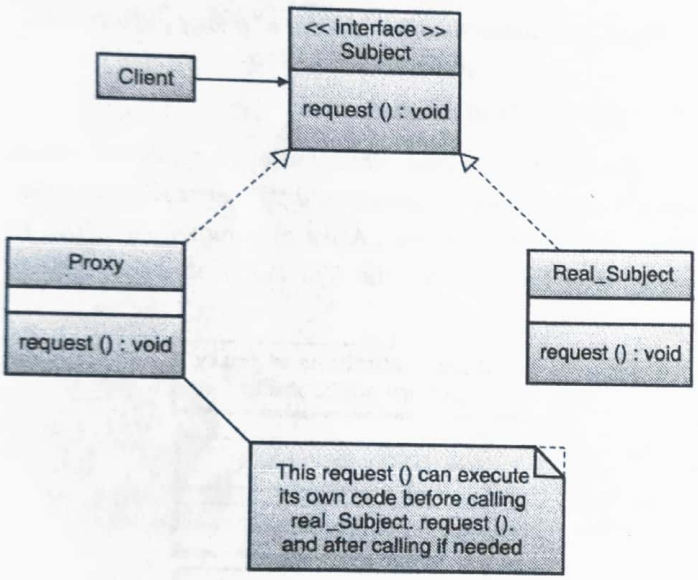


**Fig. 5.3.8**

The structure for proxy pattern is shown in Fig. 5.3.8. Different components of this structure are described in next section.

☞ **Participants of Proxy Pattern**

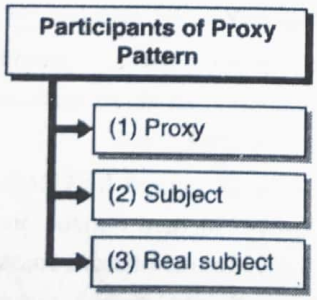


**Fig. C5.10 : Participants of Proxy Pattern**

➜ **(1) Proxy**

- Maintains a reference that helps proxy to access the RealSubject.
- Implements the same interface which a RealSubject implements. In this way a proxy can take the place of RealSubject.
- Sometimes not only controls the access to RealSubject but also can create and delete it.

➜ **(2) Subject**

This is the interface implemented by RealSubject and declares the services offered by it. This interface must be implemented by real subject and proxy.

➜ **(3) Real subject**

This is the actual object that the proxy represents.

☞ **Collaboration of Proxy Pattern**

Proxy receives the request from client, processes it and forwards it to the RealSubject.

☞ **Consequences of Proxy Pattern**

Proxy pattern provides an interface between actual object and client. It provides a level of indirection when accessing the object. This additional level of indirection has many uses. Few are listed below :

(1) The remote proxy enables a entirely different address space to be treated by client as local address space.

(2) The virtual proxy provides abstraction and optimization by creating objects on demand.

(3) Protection proxy and smart reference serves additional operations when actual object is referred.

☞ **Implementation of Proxy Pattern**

- Implementation aspect of virtual proxy will be covered in this section with the help of gallery application.
- Consider the gallery application that provides a view of photos under different section including high resolution camera images.
- As discussed previously, the high resolution images are not loaded as soon as you open the gallery instead a list is displayed of the image under different sections.
- User can select the image later to fully load the photo, render it on screen and then more operation on image can be performed.
- A class diagram applied with proxy pattern is shown in Fig. 5.3.9.

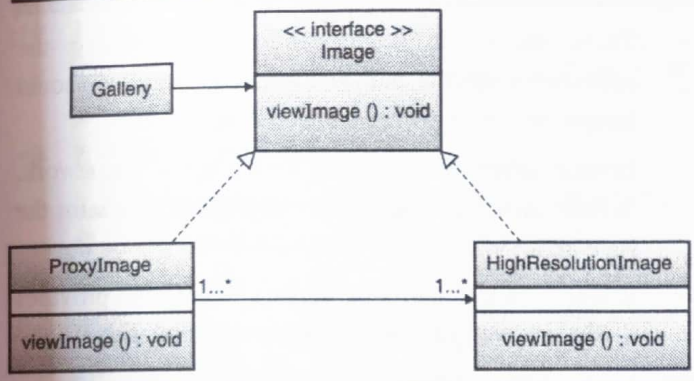


**Fig. 5.3.9 : Clean diagram for proxy pattern**

- Fig. 5.3.9 shows that both, proxy image and high resolution image implements image interface.
- When user opens Gallery application the proxy image is responsible for loading abstract image with abstract functionalities.
- viewImage() method loads and display the actual high resolution image only when it is needed.
- The code snippet shown below illustrates the Image interface which serves as subject. This interface declares an operation viewImage(). This view Image () operation must be implemented by concrete Images.

```
/* subject Interface */
public interface Image {
        public void viewImage () ;
        }
```

- The code snippet shown below is the proxy implementation. This Image Proxy class is a virtual proxy and has the responsibility of creating and loading the actual image object on demand and hence saves the cost of loading the high resolution image into memory until it is explicitly requested.

```
/ * Proxy class * /
public class ImageProxy implements Image {
        private String ImageLocation ;
        private Image ProxifiedImage ;
        / * A reference variable for RealSubject * /
        Public Image Proxy (string imageLocation) {
            this. imageLocation = imageLocation ;
}
@ override
public void viewImage () {
        / * Image loading and rendering code * /
            ------------
            ------------
}
}
```

- The code snippet shown below is the RealSubject implementation and illustrates the concrete implementation of HighResolution Image reference.
- This class is responsible for loading the high resolution image to the memory and rendering it on screen when view Image () method is called.

```
/ * Real Subject * /
public class HighResolutionImage implements
    Image {
public HighResolutionImage (String imageLocation) {
    loadImageToMemory (imageLocation)
}
private void loadImageToMemory (String imageLocation) {
    // Code for Loading image from
    // specific location in secondary storage
    // to main memory
        --------
}
@ override
public void viewImage () {
    // code for displaying the high
    // resolution image on screen
        ----------
        ----------
    }
}
```

**Syllabus Topic : Behavioral Design Pattern**

## 5.4 Behavioral Patterns

**Q. Explain behavioral pattern in short. (5 Marks)**

- Behavioral patterns deal with algorithms and assignment of responsibilities between objects.
- Behavioral pattern describes the responsibilities of object and the patterns of communication between them.
- It describes how different objects are connected with each other and what task they perform.
- Behavioral patterns mostly use object composition instead of class inheritance. It describes how different objects coordinate with each other to perform some specific task that otherwise in difficult to be performed by a single object.
- Behavioral pattern deals with the issues related to object awareness among each other. In other words, this pattern addresses the issue of how peer objects will about each other.

- Two important patterns under behavioral pattern

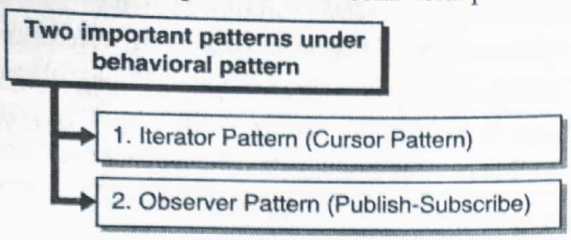


**Fig. C5.11 : Patterns under behavioral pattern**

## Syllabus Topic : Iterator Design Pattern

### 5.4.1 Iterator Pattern (Cursor Pattern)

**(Feb. 2016)**

Q. Explain characteristics, consequences and application of Observer pattern.

**SPPU - Feb. 2016(In sem), 3 Marks**

- Iterator pattern is also known as cursor pattern is used to access the element of an aggregate object in a sequential manner.

#### Characteristics of Iterator Pattern

- It allows client to iterate through the objects or object elements without exposing the internal representation of aggregate object.
- It also allows the object to be iterated in different ways, depending on what application wants to accomplish.

#### Intent of Iterator Pattern

- Iterator pattern provide the way to access the element of an aggregate object in a sequential manner and does not reveal the internal representation of aggregate object.
- Iterator pattern provides an abstraction that helps to modify the collection implementation (e.g. collection in Java) without making the changes outside the collection.

#### Motivation of Iterator Pattern

- One of the most commonly used data structure in modern programming framework is collection.
- Programming languages like java and C# uses collection framework to store and operate on group of objects. In simple words, collection is just collection of objects, for example : array, list, tree etc.
- Collection has to provide a way to access its element without exposing its internal structure.
- There should be a mechanism of traversing the collection elements and the traversing operation should be similar across different collections.
- Iterator pattern is well suited for collection framework. Iterator pattern uses an iterator object that maintains the state iteration.
- It keeps truck of currently iterated subject and provides a way to identify the next element to be iterated.

#### Applicability of Iterator Pattern

Iterator pattern can be applied in following possible scenarios :

- When it comes to access an aggregate object and its content without revealing internal representation to the object's client.
- When there is requirement multiple traversals of aggregate object.
- When there is requirement of uniform mechanism of traversal among different aggregate objects.

#### Structure of Iterator Pattern

The basic structure of Iterator pattern with the help of class diagram is shown in Fig. 5.4.1. Different participants are explained in next section.

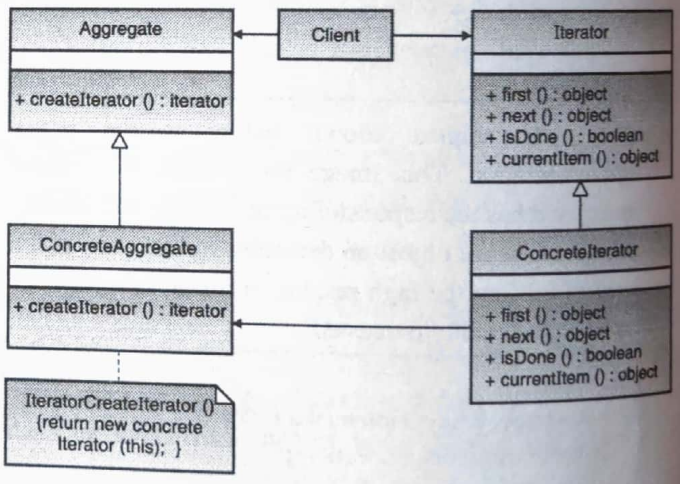


**Fig. 5.4.1 : Class diagram for Iterator pattern**

#### Participants of iterator pattern

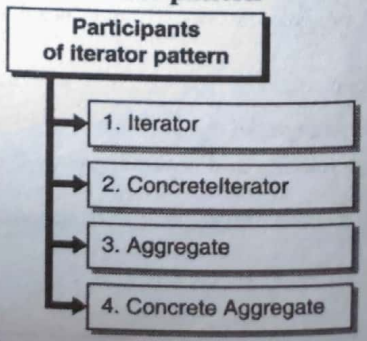


**Fig. C5.12 : Participants of iterator pattern**

➜ **1. Iterator**

Iterator defines an interface to access and traverse the element.

➜ **2. ConcreteIterator**

It is implementation of Iterator interface. It keeps track of current position in traversal of aggregate objects.

➜ **3. Aggregate**

It defines an interface for creating Iterator object.

➜ **4. Concrete Aggregate**

It is a class implementing the Aggregate interface. It return an instance of concrete iterator.

☞ **Collaboration of Iterator Pattern**

The concrete Iterator object keeps track of the current object being iterated and can decide which element needs to be iterated next.

☞ **Consequences of Iterator Pattern**

Few important consequences of Iterator pattern is listed below :

- The elements of an aggregate object can traversed in many ways. It depends on the kind of data structure and need of application that which kind of traversal mechanism is being used.
- Iterator provides a simplified interface to the aggregate object.
- In certain condition more than one traversal can be carried out on aggregate object since state of traversal can be maintained by Interator.

☞ **Implementation of Iterator Pattern**

There are many variants of Iterators and they can be implemented differently in different scenarios. Few are listed in Fig. C5.13.

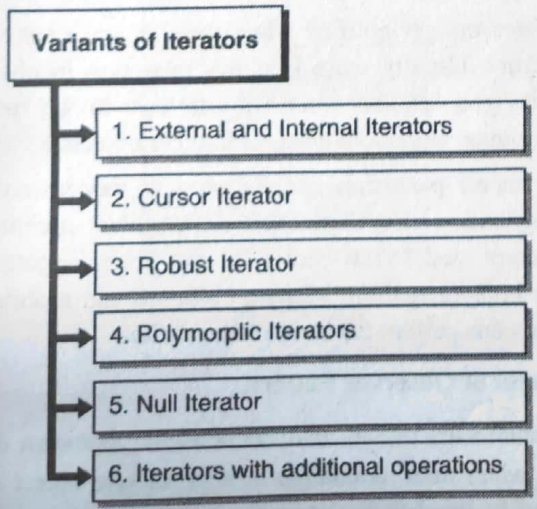


**Fig. C5.13 : Variants of Iterators**

➜ **1. External and Internal Iterators**

One important concern while implementing Iterator is "who is controlling the iteration?" When the Iterator is controlled by client, it is called **external iterator**, and when iteration is controlled by Iterator, then it is called internal iterator.

➜ **2. Cursor Iterator**

Some times iterator is just used for storing the state of iteration and actual traversal is defined by algorithm implemented by aggregate object. Such iterator is simply called as cursor.

➜ **3. Robust Iterator**

Robust Iterators are used to ensure that insertion and deletion in aggregate object should not affect its traversal. No elements should get missed or duplicated while traversing the aggregate.

➜ **4. Polymorplic Iterators**

Few languages like C++ can support polymorphic iterator. It requires object to be allocated dynamically by factory method.

➜ **5. Null Iterator**

Null Iterators are often used for handling boundary conditions a null iterator tends to always done with its operation. It implements one additional method called Done () and always evaluates to true. There iterators are useful in traversing the tree like structure.

➜ **6. Iterators with additional operations**

Few iterators are implemented with additional operations. Traditionally an iterator supports the operations like first (), next (), current item etc. But few clients and operation requires iterator to implement additional operations like skipTo (), previous (), find () etc.

## 5.4.1(A) Examples of Iterator Pattern

This is an example of phonebook contact application that simply maintains list of contacts by contact person name. The main actors are :

(1) **Iterator :** An Abstract Iterator that defines the iterator operation.

(2) **Contact Iterator :** This is implementation of Iterator. It implements I Iterator.

(3) **I container :** An interface defining the Aggregate.

(4) **Contact collection :** An implementation of collection

The class diagram for this example is shown in Fig. 5.4.2.

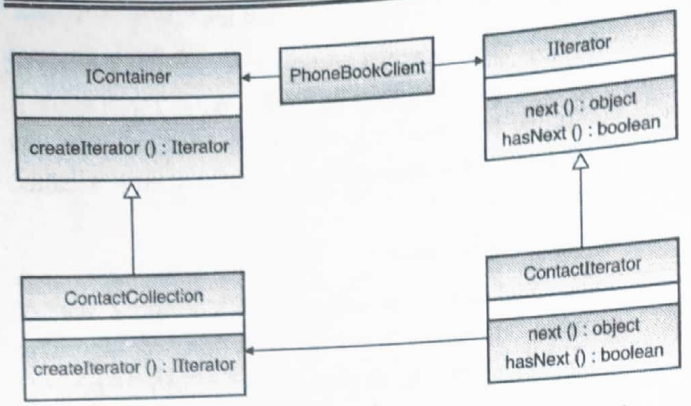


**Fig. 5.4.2 : Class diagram for phonebook application with Iterator pattern**

Code snippet for the abstraction interface IIterator is provided below. Classes which implement the IIterator interface has to provide implementation of three methods declared by IIterator.

```
interface IIterator
    {
        public object next () ;
        public boolean hasNext () ;
}
```

Similarly the interface I container has the following code implementation.

```
Interface Icontainer
{
    public IIterator createIterator () ;
}
```

The concrete class implementation for iterator and collection is provided below. Here the concrete iterator is nested class. This nesting allows it to access all members of the collection. other classes cannot access encapsulated class contact Iterator.

```
class ContactCollection implements IContainer
{
  private string p_contacts [ ] = { " Arvind" , "Bhushan",
"Chetan", "Dev" } ;
  public IIterator create Iterator ()
  {
        ContactIterator ci = new ContactIterator () ;
        return ci ;
  }
  private class ContactIterator implements IIterator
  {
     private int c_position ;
     public object next ()
     {
        if (this · has Next ())
           return c _ contacts [c _ position ++] ;
        else
           return null ;
     }
     public boolean hasNext ()
     {
        it (c _ position < c _ contacts · length)
           return true ;
        else
           return false ;
     }
  }
}
```

## Syllabus Topic : Observer Design Pattern

### 5.4.2 Observer Pattern (Publish-Subscribe)

**(Feb. 2016, May 2016)**

**Q.** Explain characteristics, consequences and application of Observer pattern.
SPPU - Feb. 2016(In sem), 3 Marks

**Q.** Explain the intent, motivation, structure, implementation of observer pattern.
SPPU - May 2016, 5 Marks

**Q.** Draw the structure of observer pattern with suitable class diagram including subject and observer. **(4 Marks)**

- Observer patterns deals with few very important aspects of object-oriented programming. It considers the object's state and object interaction.
- Observer patters is also known as publish-subscribe problem. One object is considered as subject that maintains list of dependents called observers.
- Observers get notified when there is any state change occurs. Usually there is notify operation in observers that gets called when object has to be notified automatically about changes in other objects.
- Observer pattern is heavily used in modern software application development. A popular architectural pattern model view-controller uses observer pattern for its implementation. Many web-based and mobile apps uses this pattern for its implementation.

**Intent of Observer Pattern**

Implements one-to-many dependency between objects so that when there is change in state of one object all the dependents are notified and updated automatically.

### Motivation of Observer Pattern

- Now a days there are many web-based applications and mobile application that requires observer pattern.
- Many mobile applications can be treated as observers that need to be alerted or notified for state changes in subject.
- When a system needs to be developed with many co-operating objects and the state change of one object triggers the event in other object, then designing the system by simply partitioning into classes is not wise enough.
- Consistency between objects has to be maintained but not at the cost tight coupling because you want your classes to be reusable Observer pattern provides an efficient design and implementation strategy in such scenario.
- Observer pattern specifies how to establish relationship by treating system as collection of observers and subject.
- One subject may have many dependent observers and all the observers get notified when the state of subject changes.
- It is possible that after receiving the notification, observers can query or request subject for necessary information so that they can synchronize their state with subject's new state.

### Applicability of Observer Pattern

Following are the few examples of observer pattern's application :

(1) When the change in one objects state has to be reflected in other objects, and it is not known that how many objects need to be changed every time.

(2) When objects should be able to send notification to other objects about the state change, but no prior assumption can be made about who these objects are.

### Structure of Observer Pattern

Structure of observer pattern with the help of class diagram is shown in Fig. 5.4.3 and participant is explained in next section.

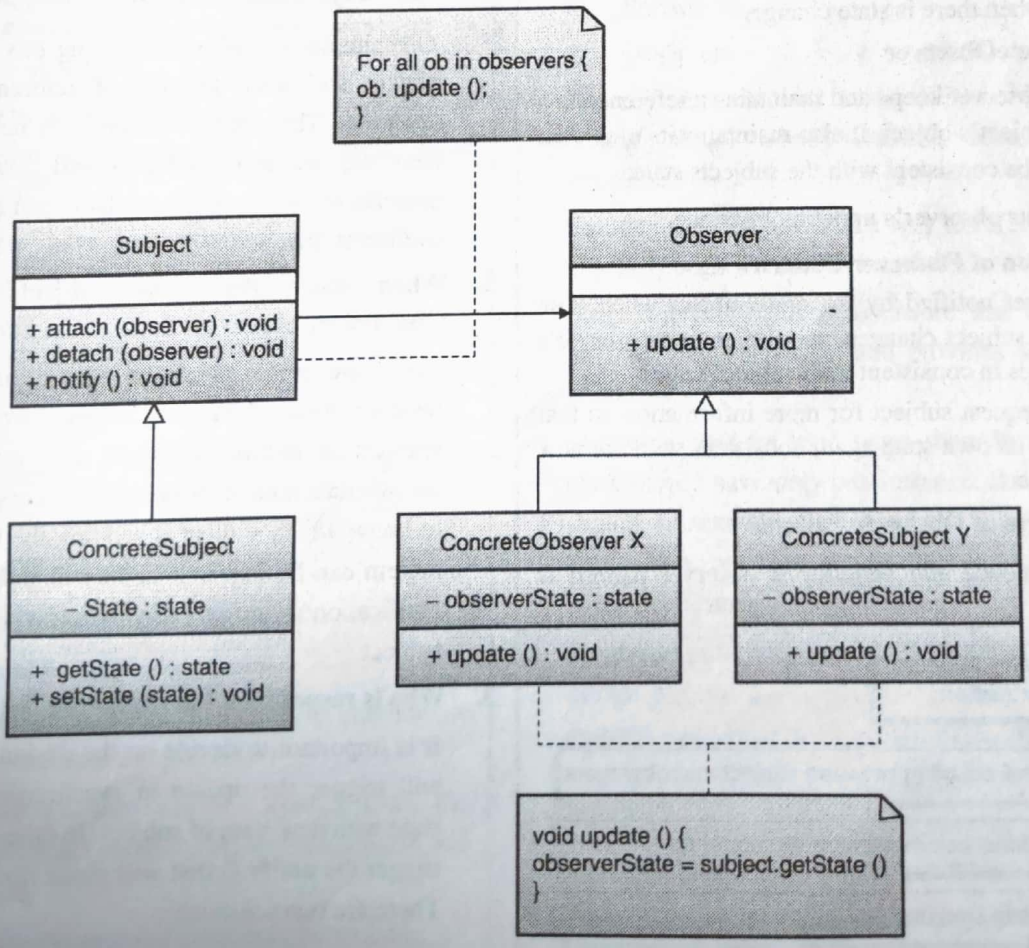


**Fig. 5.4.3 : Class diagram for Observer Pattern**

☞ **Participants of Observer Pattern**

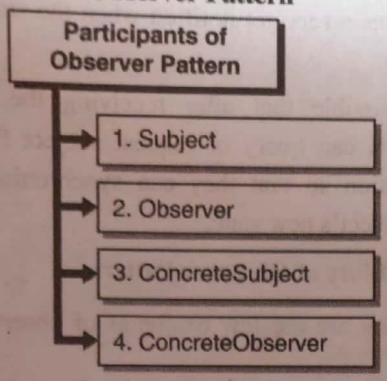


**Fig. C5.14 : Participants of Observer Pattern**

➜ **1. Subject**

Subject is observed by any number of observers for any state change. It also provides an interface for attaching and detaching observer objects.

➜ **2. Observer**

Observer declares an updating interface for objects that wants notification for change in subject.

➜ **3. ConcreteSubject**

Concrete subject stores the state that needs to be observed by observers. If sends notification to all the observers when there is state change.

➜ **4. ConcreteObserver**

Concrete Observer keeps and maintains a reference to a concrete subject's object. It also maintains its own state that should be consistent with the subjects state.

It implements observer's updating interface.

☞ **Collaboration of Observer Pattern**

- Observers get notified by ConcreteSubject when state of concrete subject changes, soon after that observer's state becomes in consistent with subject's state.
- Observers request subject for more information so that it can make its own state again consistent with the new state of subject.

☞ **Consequences of Observer Pattern**

Few consequence and benefits of observer pattern is listed.

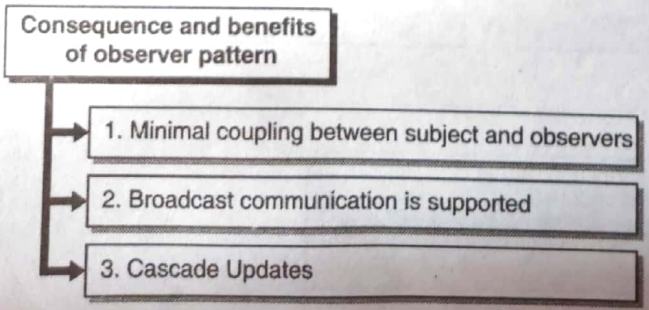


**Fig. C5.15 : Consequence and benefits of observer pattern**

➜ **1. Minimal coupling between subject and observers**

Subject known the list of observers where each observer implements the operation defined in abstract Observer class or interface. Subject however does not know the concrete class of any observer. This enables a loose coupling between subject and observer.

➜ **2. Broadcast communication is supported**

The notification need not to specify its receiver. Notification is broadcasted to all interested objects which has subscription for that state change event.

➜ **3. Cascade Updates**

It is also possible the subject's state change depends on any other object's state. In that case, the subject becomes observer and another object becomes subject. In this way the notification and updates can be cascaded to multiple levels of subject and observers.

☞ **Implementation of Observer Pattern**

There are several implementation methods and issues with observer pattern.

**1. How subject and observers are mapped together?**

One simple way of implementing this mapping is that subject will keep the list of reference to different observers. This solution however is not efficient when there are too many subjects and few observers. An associative look up like hash table can be maintained to overcome this.

**2. When more than one subjects need to be observed by observers ?**

There are some situations when observer need to observe more than one subject. For example one application notifies user about some event and depends on two data source. It becomes necessary for observer to know which subject is sending the notification such system can be implemented by slightly modifying the notification message that now includes an identifier of subject.

**3. Who is responsible for triggering the update ?**

It is important to decide on the design aspect that who will trigger the update to synchronies the observer's state with new state of subject. In other words, who will trigger the notify () that will result finally in an update. There are two solutions :

(a) One solution is provide state setting operation in subject and make a call to notify () operation once

subject changes its state. So client does not have to call notify on subject.

(b) Another solution is to give the responsibility to client so that it can call notify () at right time. The challenge is here to decide on 'right time' to call the notify () operation on subject.

4. **Handling the dangling references to deleted subjects**

One important aspect of design in this pattern is what will happen to the references of subjects in observers once the subject is deleted. There should be an implementation of mechanism in system by which observers can be notified about the removal of subject. Observers should be able to take the necessary reference updates after removal of subject.

## Summary

- **Design patterns** are well defined problem solving strategies developed over the years of experiences in software designing, it provides a reusable solution to common problems which developers encounters in software development.
- **Design pattern** is a template that can be applied in design and can be transformed into coding while implementation.
- Design patterns have four essential elements for its application, specification and description:
  a. Pattern Name
  b. Problem
  c. Solution
  d. Consequences
- Three primary classifications of patterns are:
  a. Creational Pattern
  b. Structural Pattern
  c. Behavioural Pattern
- **Creational design patterns** are used to provide an abstraction of instantiation process.
- **Creational pattern** separates the system from the complexities of object creation, object instantiation and its representation.
- Two important patterns under creational pattern are:
  a. Factory pattern
  b. Singleton pattern
- **Structural pattern** deals with formation of large structures with the help of smaller classes and objects
- **Structural patterns** are applied when the primary concern of design is to compose large structures from classes and objects.
- **Structural patterns** are often used when different developers develop individual classes and later on the classes are combined to compose more useful classes of the system.
- Two important examples of structural pattern are:
  a. Adapter
  b. Proxy Pattern
- **Behavioral** patterns deal with algorithms and assignment of responsibilities between objects.
- **Behavioral pattern** describes the responsibilities of object and the patterns of communication between them.
- **Behavioral pattern** describes how different objects are connected with each other and what task they perform.
- Two important examples of behavioral pattern are :
  a. Iterator
  b. Observer
- **Factory Pattern** used when there is requirement for creating object without revealing instantiation logic to the client.
- **Factory pattern** provides a way to refer newly created object through a common interface.
- **Singleton** pattern used to ensure that a class should have only one instance, and provides a global access point to it.
- **Singleton pattern** used when there is required that a class should have only one instance. For example there should be exactly one Window Manager in a GUI application, or there should be only one instance of Virus Scan running in the system.
- **Singleton pattern** is one of the simplest and interesting design pattern. It is exciting to observe that how it is ensured during the coding that a class should have only are object at a time.
- **Adapter** class works like adapter in real world and serves as an interface or bridge between two objects.
- **Adapter pattern** converts interface of a class into another interface that client expects.

- **Adapter** provides a way for incompatible class to work together.
- **Adapter pattern** is applied when there is need of wrapping an existing class with new Interface.
- The dictionary meaning of **adapter** is "Device that enables something to be used in a way different from that for which it was intended" or " which makes different pieces of apparatus compatible" This pattern encourages the use of such adapter class that provides adapting between classes and objects.
- In general proxy is a class functioning as an interface to something else. Proxy can provide an interface to things like : a file, a network connection or any other expensive resource which is hard to duplicate.
- The intent of **Proxy pattern** is to provide a surrogate or placeholder for another object to control access to it.
- **Proxy** can be considered as an agent or wrapper that communication to the client and provided an abstraction of real serving object behind the scenes.
- **Proxy pattern** is applied when there is need of controlled access to an object. The reason of controlling access to an object in to temporarily delaying or deferring the full cost of creation and initialization.
- **Iterator pattern** is also known as cursor pattern is used to access the element of an aggregate object in a sequential manner.
- **Iterator pattern** allows client to iterate through the objects or object elements without exposing the internal representation of aggregate object.
- **Iterator Pattern** also allows the object to be iterated in different ways, depending on what application wants to accomplish.
- **Iterator pattern** provide the way to access the element of an aggregate object in a sequential manner and does not reveal the internal representation of aggregate object.
- **Iterator pattern** provides an abstraction that helps to modify the collection implementation (e.g. collection in Java) without making the changes outside the collection.
- **Observer patterns** deal with few very important aspects of object-oriented programming. It considers the object's state and object interaction.
- **Observer** patters is also known as publish-subscribe problem. One object is considered as subject that maintains list of dependents called observers.
- **Observers** get notified when there is any state change occurs. Usually there is "notify" operation in observers that gets called when object has to be notified automatically about changes in other objects.
- **Observer pattern** is heavily used in modern software application development. A popular architectural pattern model view-controller uses observer pattern for its implementation. Many web-based and mobile apps use this pattern for its implementation.

## 5.5 Exam Pack (University and Review Questions)

Q. What are design pattern and why they are important in modern software development. ?
*(Refer section 5.1.1)* (4 Marks)

Q. Explain the essential elements of design pattern. *(Refer section 5.1.2)* (4 Marks)

Q. What are the major classification of design pattern? Explain in brief with few example patterns under each class. *(Refer section 5.1.3)* (6 Marks)

Q. What are the major elements of documenting and describing the design pattern?
*(Refer section 5.1.4)* (5 Marks)

☞ **Syllabus Topic : Introduction to Creational Design Pattern**

Q. What is creational pattern ?
*(Refer section 5.2)* (2 Marks)

☞ **Syllabus Topic : Factory**

Q. Write short note Factory Pattern.
*(Refer section 5.2.1)* (5 Marks) **(Feb. 2016(In Sem.))**

Q. Explain factory pattern. Describe its intent, motivation and implementation with suitable example.
*(Refer section 5.2.1)* (8 Marks)

☞ **Syllabus Topic : Singleton**

Q. Describe Singleton pattern and its significance with an example. *(Refer section 5.2.2)* (5 Marks)
**(Feb. 2016 (In Sem.))**

Q. Explain the intent, motivation, applicability, implementation and consequence of singleton pattern.
*(Refer section 5.2.2)* (6 Marks)

Q. What is singleton pattern ? Explain one example scenario where you will singleton pattern to get applied. *(Refer section 5.2.2)* (6 Marks)

### Syllabus Topic : Structural Design Pattern

Q. What is structural pattern?
*(Refer section 5.3)* (2 Marks)

### Syllabus Topic : Adapter Design Pattern

Q. Explain the intent, motivation, applicability, implementation and consequence of Adapter Pattern. *(Refer section 5.3.1)* (6 Marks)

### Syllabus Topic : Proxy Design Pattern

Q. Write short note proxy pattern.
*(Refer section 5.3.2)* (5 Marks) **(Feb. 2016 (In Sem))**

Q. Explain the intent, motivation, applicability, implementation and consequences of proxy pattern. *(Refer section 5.3.2)* (8 Marks)

### Syllabus Topic : Behavioural Design Pattern

Q. Explain behavioral pattern in short.
*(Refer section 5.4)* (5 Marks)

### Syllabus Topic : Iterator Design Pattern

Q. Explain characteristics, consequences and application of Observer pattern. *(Refer section 5.4.1)* (3 Marks)
**(Feb. 2016(In Sem.))**

### Syllabus Topic : Observer Design Pattern

Q. Explain characteristics, consequences and application of Observer pattern. *(Refer section 5.4.2)* (3 Marks)
**(Feb. 2016(In Sem.))**

Q. Explain the intent, motivation, structure, implementation of observer pattern.
*(Refer section 5.4.2)* (5 Marks) **(May 2016)**

Q. Draw the structure of observer pattern with suitable class diagram including subject and observer.
*(Refer section 5.4.2)* (4 Marks)

❑❑❑

UNIT VI

CHAPTER 6

# Testing

**Syllabus Topics**

Introduction to testing, Error, Faults, Failures, verification and validation, White Box Testing, Black Box Testing, Unit testing, Integration testing, GUI testing, User acceptance Validation testing, integration testing, scenario testing, performance testing. Test cases and test plan. Case studies expected for developing usability test plans and test cases.

**Syllabus Topic : Introduction to Testing**

## 6.1 Introduction to Testing

**Q. Explain the basic concept of software testing and its importance. (4 Marks)**

- Today software systems have become very influential in every sector and industry. Even in our home, we are surrounded by devices that use software application in some form. The revolution of digitizing every service has increased development rate of software system dramatically. They are now getting produced at highest rate ever.
- Despite of this high volume of software development and dependency of almost every sector on software, no one can compromise with the quality of software being used or developed. No organization will wish to release its product without ensuring that the product is quality product. No customer will wish to use any software product that is not properly tested. The rise of need for software everywhere has given the rise of need for testing at every stage. Software organizations are trying even harder to ensure that they are releasing defect free software because of many reasons like: very high competition in delivering quality product, fast delivery cycles and update releases, commitment towards valued customer to deliver only quality products.
- Another major reason of having well tested product is that large number of users now using the same product making it almost impossible for even smallest defect to remain undetected. Organizations have to consider every possible use case scenario while testing, even the scenario that they have never documented or realized during development.
- A well tested product requires a well defined testing methodology in place. A wide range of testing methodologies is available, covering different aspects of system. Just like design methodologies testing methodologies are also equally important in successful system implementation. Like designing and coding, testing also creates scope for creativity and innovation.
- Designing test case has become ever challenging. Test engineers have started applying research methodologies in testing process. We will discuss few major and interesting testing methodologies in this chapter. Before that we will try to define software testing and will understand the concept of testing.
- Software testing is the process of evaluating system and its components to ensure that it satisfies the all specified requirement and there is no defect in the system.
- Testing includes system execution with the intent to find any error, unfulfilled requirement, gaps, and bugs

in the system. During execution one or more test properties are evaluated.

- Following are the few indicators which are verified during the testing process.
  - System meets the requirement that guided the software design and implementation.
  - System is responding appropriately to all kinds of possible inputs.
  - System is performing the expected task within the permissible and acceptable time.
  - System is as usable as it has been expected.
  - System can be installed and run smoothly in expected environment
  - System meets the requirements and achieved the result stakeholders expected.
- It should be noted that number of all possible tests that can be performed in even a small piece of software can be practically infinite. The challenge is to decide which tests are feasible with available time and resource.
- Effective and timely testing increases the chance of a software product or the software service meeting the client's requirement. Zero defects cannot be guaranteed but effective testing definitely increases the chances of client's acceptance of software.
- Earlier there were assumptions that testing is performed right after the coding finished or when we get the software ready in executable form. But with changed development trends, when and how testing will be performed is decided by software development approach. For example iterative development requires an iterative approach in testing, where after first iteration of design, development and testing, another cycle starts. Similarly, Agile approach requires concurrent development and testing.

### 6.1.1 Principles of Software Testing

→ (Dec. 2016)

**Q.** State and explain different software testing principles of testing. **SPPU - Dec. 2016, 6 Marks**

We have briefly explored the importance, role and concept of software testing in previous section. There are few principles of testing it helps to fix the objectives of testing. These fundamentals are as follows :

1. The primary and broad objective of testing is to find the defects in software before customer finds them.
2. Covering all possible testing is not possible for even the smallest software system. In other words, exhaustive testing is very difficult or mostly not possible. We can find the presence of defect but can never guarantee absence of them.
3. Testing was earlier assumed as an end of development cycle activity. But in modern days of software development, testing is applied in all stages throughout the development life cycle of software.
4. Testing requires that the objective of testing should be clearly defined before starting the test.
5. Test methodologies and process should be clearly understood and verified before starting the test.
6. Test engineer should never stick to fixed framework of testing. They need to think out of the box for designing their test cases and conditions. Testing need to be revised constantly.
7. Defects may have a fixed pattern. They may occur in clusters. Testing should be aimed to identify the pattern and should focus on clusters.
8. Testing also include measures for defect prevention. One cycle of testing results in an improved version of software. Identified defects are removed and system is tested again for integrity.
9. Defect prevention and defect detection are two important aspects of system that testing addresses. Testing maintains an appropriate balance between them.
10. Testing is equally challenging as software designing and implementation. It requires the innovation and strategic approach. It is one of the interesting fields of computer science and requires research methodologies to make it more efficient.

### 6.1.2 Testing Strategies

→ (May 2016)

**Q.** Explain different testing strategies. **SPPU - May 2016, 6 Marks**

The term "Testing Strategies" is commonly used to refer different kinds of testing. We will be discussing several kinds of testing in the subsequent sections of this chapter. Before that we will try to have more formal understanding of the term in this section.

Testing strategy is set of procedures, specifications that describe software testing approach in software development life cycle. An organization can define its testing strategy

uniformly across organization or specifically for ongoing project. All the stakeholders in a development project like developers, test engineers, system analysts, business analyst and customer or clients should ideally be aware of the testing strategy adopted by the project management.

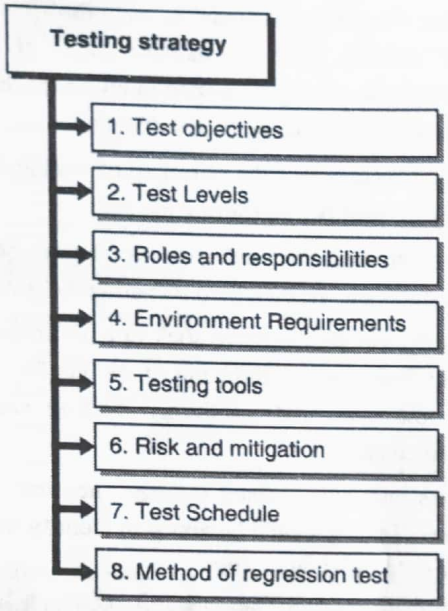


**Fig. C6.1 : Testing Strategy**

### ➜ 1. Test Objective

- High level objective of testing is common to all organization. It is to find the defect in the system. Other objectives can be defined and prioritized by different organization as per their requirements and policies.
- More or less, the primary objectives however are same for all software development companies. They are listed below :
  - Finding the defects in the system which may get created during the system implementation
  - Gaining the confidence while providing the information about the level of quality
  - To ensure the prevention of defects
  - To ensure the end product meets the users or business requirement
  - To earn the confidence of customer by providing them only quality product

### ➜ 2. Test Levels

- Test strategy defines the test level to be performed. There are four primary levels of test :
  (a) Unit Testing level
  (b) Integration testing level
  (c) System testing level
  (d) Acceptance testing level
- Unit test is mostly performed by the developers while integration and system testing is performed by testing team. Business analysts and customers can participate in acceptance testing.
- Test levels are shown in Fig. 6.1.1 where testing is performed using bottom up approach.

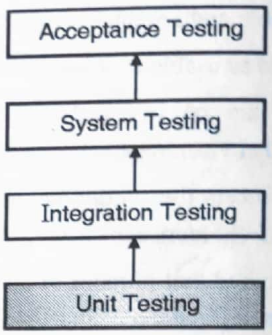


**Fig. 6.1.1 : Levels of testing : A bottom up approach**

### ➜ 3. Roles and responsibilities

- Roles and responsibilities of the participants in testing activity are clearly defined as part of testing strategy.
- It defines the roles of developers, quality analysts and test engineers in different stages of testing.
- Roles and responsibility may vary from organization to organization depending on type of life cycle model or on other factors.
- **Example :** V model requires active participation of developers and architects in test designing.

### ➜ 4. Environment requirements

- Testing strategy also describes the environment in which testing is going to be performed.
- It includes hardware and operating system details. It may include additional details like necessary security updates on the platform and operating system patches.

### ➜ 5. Testing tools

- Many organizations are now switching from manual testing to combination of manual and automated testing.
- Organization are now standardizing the automated testing process by training the developer and test engineers on automated tools.

**6. Risk and mitigation**

- Testing strategy also includes the description of risk associated with testing strategy being used along with mitigation.
- The risk anticipation provides scope for proactive action to be taken in advance.

**7. Test Schedule**

- One of the important part of testing strategies is the details of the testing schedule.
- It is often called the test plan that specifies how long it will take complete each phase of testing.

**8. Method of regression test**

- Testing strategy specifies the methods of ensuring that testing has successfully removed the defect without any side effect.
- Regression tests are used to ensure that system is working properly after enhancement or patch been applied.

**Syllabus Topic : Error, Faults and Failure**

## 6.2 Error, Faults and Failure

Three important terms associated with software that must be understood before going into details of software testing are: error faults and failure. These are three related but distinct terms in software testing. Let us discuss the terms in brief.

### 6.2.1 Error

**Q. Write short note on error in software testing. (4 Marks)**

- **Error** : Human mistake or the human action that produces the incorrect result is called error.
- This can be programming error, design error, architecture level error or even testing errors. Sometimes error is caused by external factors too.
- In more traditional terms, often the amount of deviation from expected result and actual result is called error. This definition of error can also be applied in software errors.
- Errors can be of several types in software like: design error, documentation error, coding error, architectural error, functional error, hardware defects, requirement errors, performance errors, interface errors etc.

### 6.2.2 Faults

**(Dec. 2015)**

**Q. Define fault. SPPU - Dec. 2015, 2 Marks**

- **Faults** : The appearance of the error in software is called fault. This is also known as manifestation of error in software. Faults are also known as bug or defect.
- Fault is often considered as state of software which is caused by an error.

### 6.2.3 Failure

**(Dec. 2015)**

**Q. Define failure. SPPU - Dec. 2015, 2 Marks**

- **Failure** : The deviation of a software product from its expected behavior due to faults is called failure. In the case of failure, a software system fails to deliver the expected service.
- Failure is directly observable to the external users. Failure signifies that the external behavior of the system is incorrect.
- Failure can also be caused by environmental or situational defects.

**Syllabus Topic : Verification and Validation**

## 6.3 Verification and Validation

**Q. Explain verification and validation of software system. (6 Marks)**

Conventional definition of testing says "Testing is the phase in software development life cycle that follows coding and precedes deployment". In modern software development, however coding may continue after one phase of testing or sometime continues side by side with testing. Traditionally testing is used to refer the testing of code but in practice the design and requirements can also be tested. For testing only program code is not sufficient. Defects may get injected in software design or even in requirement specification. In such scenario, delays testing till completion of coding will incur significant resource loss. Two important terms need to be explored to understand the ways of quality improvement and defect detection :

1. Verification 2. Validation

- Testing is not only the means for detecting the defect and ensuring quality of the system. There could be other methods like quality review process and formal

inspections of the system. There are two important processes that allow us catch the defect and improve the quality: Verification and Validation

- Verification and validation also known as V&V in software testing, is the process of checking that software system meets the required specification and fulfils all expectations. These are generally assigned as task in software testing and test engineers are often responsible for performing validation and verification.
- **Verification** is the process of evaluating a software system and its components to determine whether the product for given phase satisfies the conditions imposed at beginning of that phase.
- **Validation** is the process of evaluating a system and its components during or at the end of the development process to determine whether it satisfies overall requirements.
- The verification process is concerned with the activities which addresses the question "Are we building the product right?"
- Similarly validation is set of activities which are concerned with answering the question "Are we building the right product?"
- **Building the product right (Verification) :** This includes certain activities and conditions that are imposed at the starting of software development life cycle. These activities are often termed as *proactive* because the objective is to prevent the defect before they occur. These activities can be performed during different phase of software development and are called verification.
- **Building the right product (Validation) :** This includes certain activities which are performed during various phase to validate that whether the product is built as per specification. These activities are often termed as *reactive* because the objective is to find the defect in product at hand and fix it as soon as it is introduced. Unit testing, integration testing, system testing are few example.
- Often the term **quality assurance** and **verification** is taken in same context. Similarly the term quality control, validation and testing are taken into same context.

Quality Assurance = Verification

Quality Control = Validation, Testing.

- Fig. 6.3.1 illustrates the verification and validation in software development life cycle.
- The overall system requirements and problem definition is fixed at the beginning that exactly specifies what system need to be built. Developers and test engineers keep on referring these specifications for *validation*.
- On the other hand, procedures and preconditions are formulated before software development lifecycle that ensures the quality product being developed. System is ***verified*** against these specifications, procedures and conditions at different stages in development and implementation.

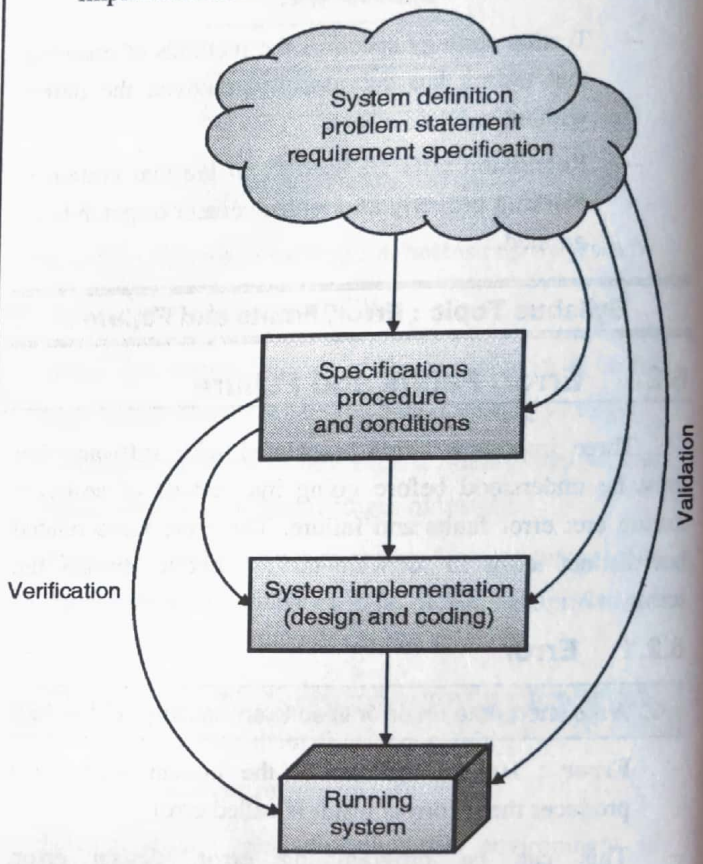


**Fig. 6.3.1 : Verification and validation overview**

## 6.3.1 Difference between Validation and Verification

➔ **(May 2016)**

**Q.** Explain the difference between software verification and validation with an example.

**SPPU- May 2016, 6 Marks**

- More specific difference between validation and verification is provided in the Table 6.3.1.

**Table 6.3.1**

| Sr. No. | Parameters | Validation | Verification |
|---|---|---|---|
| 1. | Approach | Validation addresses the question: Are we building the right product? | Verification addresses: Are we building the product right? |
| 2. | Objective | Validation is concerned with customer's actual need. | Verification checks whether the system is well-engineered, rightly processed and error free. |
| 3. | Usefulness | Validation ensures usability of system to the customer. It checks whether system fulfils customer's need. | Verification ensures the system is of high quality but does not necessarily ensure whether it useful to the customer |
| 4. | Process | Validation is subjective process. It includes subjective assessment of the system to ensure that system meets the real world requirements. | It is more objective process. Verification includes activities that results in high quality software product. This is an activity based process. No subjective assessments are needed. |
| 5. | Methods | Validation includes subjective assessment methods like requirement modeling, prototyping, walkthrough, user evaluation etc. | Verification includes activities like testing, design analysis, system and component inspection etc. |
| 6. | Applicability | Traditionally, validation is applied only at beginning and ending of project. However, modern systems with changing requirements apply validation at each phase. | Verification checks the product in each phase of software development life cycle. It checks whether the product meets the requirement for the current phase. |
| 7. | Orientation towards quality | Validation is associated with quality control | Verification is associated with quality assurance |
| 8. | Quality View | Validation is considered as customer's view of quality | Verification is considered as producers view of quality |
| 9. | Execution View | Validation does not necessarily require software to be executed. | Verification requires the execution of software. |
| 10. | Testing Technique | Validation may use static testing techniques. | Verification uses dynamic testing techniques. |

- A range of verification and validation techniques are available and different software organization can pick from these techniques based on the nature of software product and available resources.
- The set of techniques is often termed and V and V toolbox.
- Fig. 6.3.2 shows these sets of techniques.

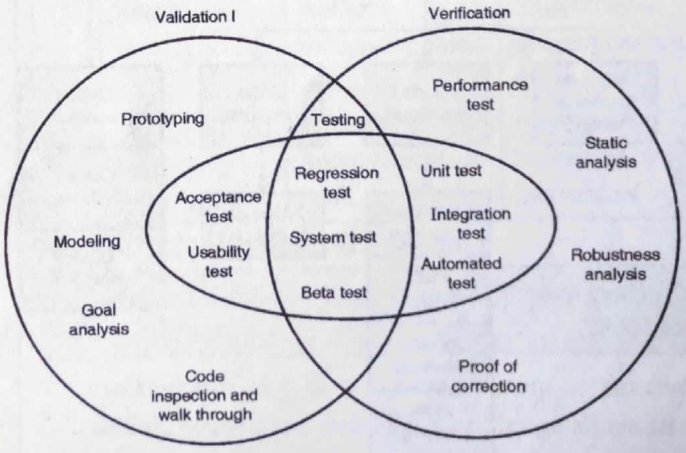


**Fig. 6.3.2 : V and V Techniques**

## 6.4 White Box Testing and Black Box Testing

There are two widely known and popular classification of testing called white box testing and black box testing. We will explore both the testing approaches in this section one by one.

**Syllabus Topic : White Box Testing**

## 6.4.1 White Box Testing

**Q. Write short note on white box testing. (5 Marks)**

- White box testing tests internal structure and working of a system. It checks the external functionality by analyzing the code of system.
- White box testing is also called glass box testing or clear box testing or open box testing.
- Internal structure and code of the system is tested and then realized into external functional of the system.
- Most of the defects detected in white box testing is due to improper logic or due to improper transformation of design and requirement into program code.
- White box testing is classified into two categories of testing : Static testing and structural testing.
- Classification of white box testing : static and structural testing along with their sub types is shown in Fig. 6.4.1.

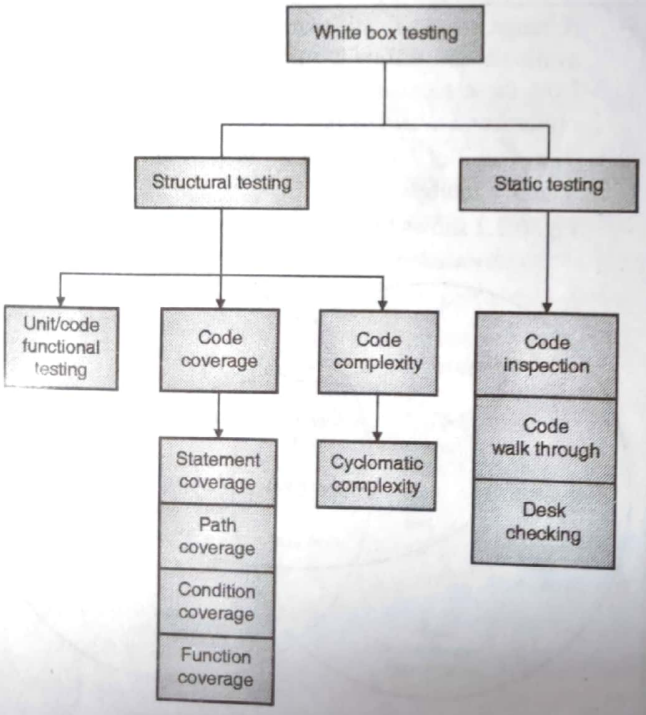


**Fig. 6.4.1 : Classification of white box testing**

☞ **Classification of white box testing**

- Two broad classification of white box testing is : Static testing and structural testing. We will discuss, their two testing in next sections.

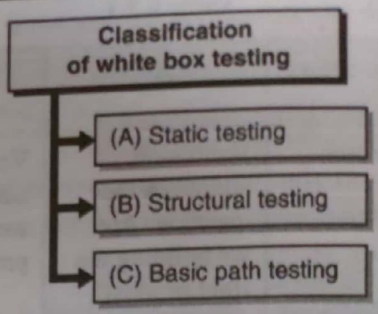


**Fig. C6.2 : Classification of white box testing**

### ➜ 6.4.1(A) Static Testing

**Q. What is static testing? (2 Marks)**

- Static testing is a type of white box testing where only source code is required for testing. Binary and executable files are not considered.
- Program or codes are not executed in static testing. Only code is examined by selective people.
- Source code in examined in static testing for finding the following facts :
  (a) Code is handling all the errors and exception.
  (b) Code has been written with the specified convention and guidelines.
  (c) Code is written as needed for the functional requirements.
  (d) Code is the exact implementation of design developed in previous phase of life cycle.
  (e) Every functionality is covered in code.
- Static testing is mainly performed by human and generally are not automated.

☞ **Different methods of static testing**

**Q. State and explain different methods of static testing. (4 Marks)**

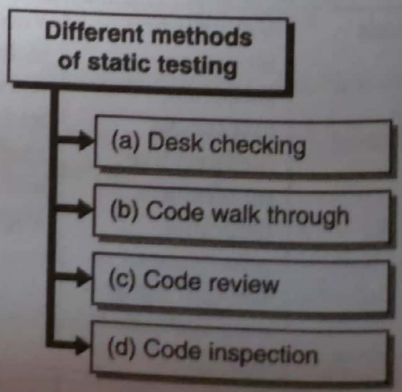


**Fig. C6.3 : Methods of static testing**

➜ **(a) Desk checking**

- After writing the code, author verifies the code for correctors.
- Author of code can verify it by comparing with required specification or design.
- This is done before compiling and executing the code.

➜ **(b) Code walk through**

- This is slightly more formal way of testing the code where one assigned person, probably the project manager or senior software engineers sits with developer/author and tries to understand the code which are not clear.
- They can ask question to the developer regarding code, style conventions and logic, if not answered, the code has to be reviewed or changed.

➜ **(c) Code review**

- This even more formal way of code testing where questionable code is submitted for formal review by experts.
- The review is documented and feedback is provided to the author of code.
- This is the systematic examination of code.

➜ **(d) Formal inspection**

- This is the most formal way of code inspection and testing.
- Here the primary objective is detect the fault, violations and side effects.
- Formal inspection starts only when author gets the code checked form desk checking, code walk through and code review.
- Authors are present but not involved in code inspection directly.
- Inspection is performed a formal committee of expert code inspectors.

☞ **Difference between code walk through and formal inspection**

**Q. Compare walk through and formal inspection. (5 Marks)**

| Sr. No. | Parameters | Code Walkthrough | Formal Inspection |
|---|---|---|---|
| 1 | Method | Code walkthrough is less formal method comparing to Formal inspection | This is strictly a formal method code inspection |
| 2 | Involvement | Authors actively involved in walkthrough process and answers various question during the process | No active participation of code author |
| 3 | Approach | This is not pre-planned and systematic | This is systematic and all the schedules are planned for inspection |
| 4 | Consequences | Consequences do not affect the project. | Consequence may affect the project and major changes might be needed. |
| 5 | Result | Authors informally notes down the suggestions and improvements in code. | Suggestions, instructions are formally recorded and redirected by moderators. |

## ➜ 6.4.1(B) Structural Testing

**Q. What are different approaches of structural testing ? (6 Marks)**

- Another category of white box testing is structural testing, where code, code structure, design all are taken into consideration. Here code is executed and then behavior is observed for written code and design of system.
- Test cases are designed and applied for different parts of system.
- More and more code are tried to be executed under the designed test cases.

☞ **Three sub types of structural testing**

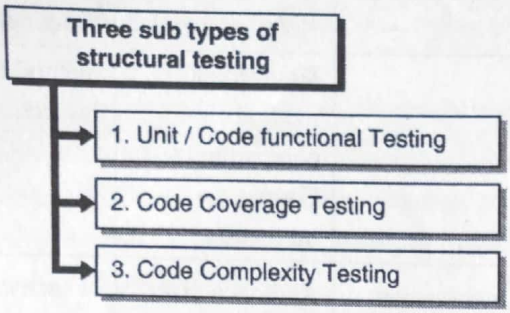


**Fig. C6.4 : Three sub types of structural testing**

➜ **1. Unit/Code Functional Testing**

- This is the most basic initial phase of structural testing where developer performs a quick check of code units and functions by running the code.
- This is performed before more detailed coverage testing.
- Developer can use debugger software to run the code and observe the behaviour of application. Developer checks the smaller set of input code and validates the output.

➜ **2. Code Coverage Testing**

- Different parts of the code are tested and examined in detail in code coverage testing.
- Code coverage testing activity includes designing and executing the test cases so that, as much as possible code can be covered under test case execution.
- 'Instrumentation' of code is a technique to find the amount of code covered by testing. Tools are available which can conduct 'instrumentation' of code and can support code coverage testing.

☞ **Types of code coverage**

- Code coverage testing is composed of following types of coverage.

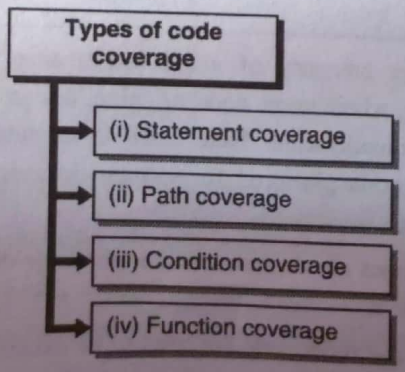


**Fig. C6.5 : Types of code coverage**

➜ **(i) Statement coverage**

- Statement coverage involves writing test cases that can cover and execute each of the program statement.
- It is always desired that program statements should be covered as much as possible.
- If more and more codes are covered, there will less possibility of defect detection after structural testing.
- Statement coverage can be stated as :

$$\text{Statement coverage} = (\text{Total statement exercised} / \text{Total number of executable statement in program}) \times 100$$

- The nature of statements can be generally classified as,
  - (a) Sequential control flow statement
  - (b) Two way decision statement (e.g. if then else)
  - (c) Multiway decision statements (e.g. switch)
  - (d) Loop statements. (e.g. for, while, do while)
- There are separate ways of designing test case for each kind of statements.

➜ **(ii) Path coverage**

- In path coverage testing, the program is splitted into number of different possible paths and test cases are written to cover each path.
- We will explore basis path testing separately in section 6.4.1(C).

➜ **(iii) Condition coverage**

- All path coverage does not ensure that entire program is covered for testing.
- The condition variable can take wide range of values. Condition expression can involve many arithmetic operators that results in different values.
- Condition coverage ensures inclusion of all possible conditions in the program.

➜ **(iv) Function coverage**

- Function coverage is concerned with covering all the program functions in the application.
- Test cases are written so that all the available functions will be called atleast once in the applications.
- It includes verification of function arguments and return values code complexity testing.

➜ **3. Code Complexity Testing**

**Q. Explain code complexity testing and cyclomatic complexity of code with the help of suitable example. (4 Marks)**

- Code complexity testing is a type of white box testing which is conducted to measure and test the complexity of code.
- This type of testing uses a software metric called cyclomatic complexity to measure the complexity of program.
- It measures the number of linearly independent paths in a program source code.
- The program is represented as flow graph consisting of nodes and edges. A program in first written as standard flow chart and then converted into flow graph.
- We will take an example small program and will draw the corresponding flow chart.
- Then will try to convert the flow chart into a flow graph and then will calculate the cyclomatic complexity for the given program code.
- Consider the following code.

```
if x = 100 then
    if y > z then
        x = y
    else
        x = z
    end if
end if
print x
print y
print z
```

- For the above code snippet, the flow chart can be drawn as shown in Fig. 6.4.2(a)

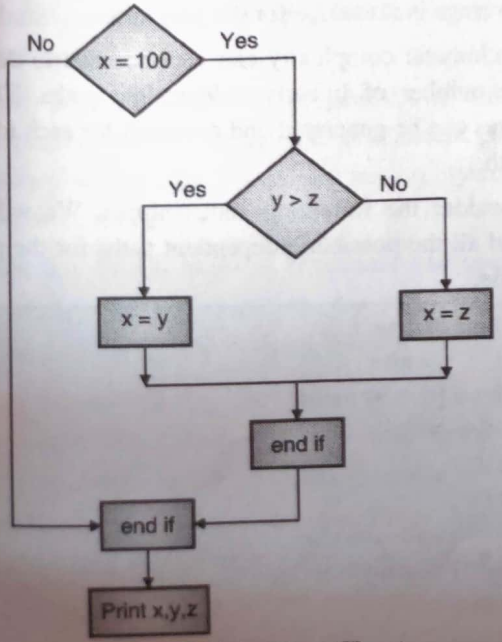


**Fig. 6.4.2(a) : Flow Chart**

- The flowchart can be converted into a flow graph by identifying the predicate node (decision node) and by combining all sequential statements into a single node. The flow graph can be drawn as shown in Fig. 6.4.2(b).

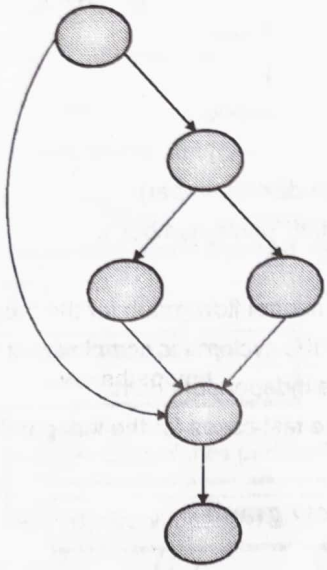

**Fig. 6.4.2(b) : Flow graph**

- Now the cyclomatic complexity of program code can be calculated as :

Cyclomatic complexity = E – N + P

Where E = number of edges in the flow graph
N = number of nodes in the flow graph
P = number of predicate node (or the decision nodes)

- So, for the flow graph, shown in Fig. 6.4.2(b), cyclomatic complexity can be calculated as,

Cyclomatic complexity = E – N + P = 7 – 6 + 2 = 3

- Cyclomatic complexity can also be calculated by calculating the number of predicate nodes and adding one in it.

So,

Cyclomatic complexity = number of predicate nodes + 1 = 2 + 1 = 3

**Ex. 6.4.1** **SPPU- May 2016, 10 Marks**

Consider the following program segment.

```
Main()
{
        Int number, index;
        Printf ("Enter a number");
        Scanf("%d",&number);
        Index=2;
```

```
        While (index<=number-1)
        {
                If(number%index==0)
                {
                        Printf("Not a prime number");
                Break;
                }
                Index++;
    }
        If (index==number)
        Printf("prime number");
    }
```

i) Draw the control flow graph for the program.

ii) Calculate the cyclomatic complexity of the program.

iii) List all the independent paths.

iv) Design the test cases for the independent path.

**Soln. :**

**(i) Control flow graph**

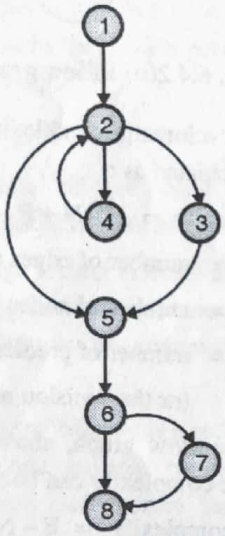


**Fig. P.6.4.1**

**(ii) Cyclomatic complexity**

$$V(G) = E - N + 2 = 10 - 8 + 2 = 4$$

**(iii) Independent paths**

Path (1) 1 – 2 – 4 – 2 – 3 – 5 – 6 – 8

Path (2) 1 – 2 – 4 – 2 – 3 – 5 – 6 – 7 – 8

Path (3) 1 – 2 – 4 – 2 – 5 – 6 – 8

Path (4) 1 – 2 – 4 – 2 – 5 – 6 – 7 – 8

Path (5) 1 – 2 – 5 – 6 – 8

Path (6) 1 – 2 – 5 – 6 – 7 – 8

Path (7) 1 – 2 – 3 – 5 – 6 – 8

Path (8) 1 – 2 – 3 – 5 – 6 – 7 – 8

**(iv) Test cases**

**Case 1**

Number entered = 0

Path (5) verified

**Case 2**

Number entered = 2

Path (6) verified

**Case 3**

Number entered = 3

Path (4) verified

**Case 4**

Number entered = 4

Path (7) verified

**Case 5**

Number entered = 5

Path (4) verified

And so on ....

## ➜ 6.4.1(C) Basic Path Testing

- Basis path testing is a white box method for designing and conducting the test. This method tries to examine and analyze the control flow graph of a program to find linearly independent path of instruction execution.
- The main objective of basis path testing is to ensure that all path of execution is examined at least once.
- If all possible paths are covered during test execution, it can be stated that statement coverage and branch coverage is also done for the program.
- Cyclomatic complexity can also be used to determine the number of linearly independent paths. Then test cases can be generated and executed for each identified path.
- Consider the following code snippet. We will try to find all the possible independent paths for the program code.

```
1. position = size + 1
2. for i = 1 to size
3. if (array [i] = = value)
4. position = i;
   end if;
   end for ;
5. for i = position to size
6. array [i] = array [i + 1];
   end for;
7. size - - ;
```

- For above code snippet, flow graph can be drawn as shown in Fig. 6.4.3.

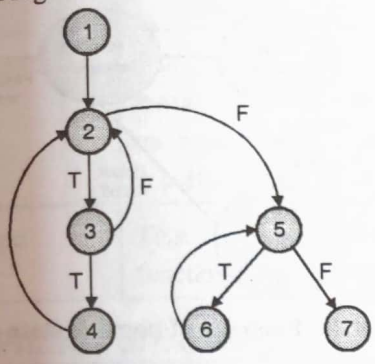


**Fig. 6.4.3 : Flow graph for program code**

- Now the independent paths can be clearly observed and listed. Test cases can be designed for basis path testing based on the following paths :

Path 1 : $1 \rightarrow 2 \rightarrow 5 \rightarrow 7$

Path 2 : $1 \rightarrow 2 \rightarrow 5 \rightarrow 6 \rightarrow 5 \rightarrow 7$

Path 3 : $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 5 \rightarrow 7$

Path 4 : $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 5 \rightarrow 6 \rightarrow 5 \rightarrow 7$

Path 5 : $1 \rightarrow 2 \rightarrow 3 \rightarrow 2 \rightarrow 5 \rightarrow 7$

Path 6 : $1 \rightarrow 2 \rightarrow 3 \rightarrow 2 \rightarrow 5 \rightarrow 6 \rightarrow 5 \rightarrow 7$

## Syllabus Topic : Black Box Testing

### 6.4.2 Black Box Testing

**Q. What is black box testing ? (4 Marks)**

- Black box testing involves testing methodologies which examine the functionality of an application or system without peering into the program code and internal structure.
- Black box testing is performed from customer's perspective and can be applied at different phases of software testing like : unit, integration, systems and acceptance testing.
- Black box testing requires functional and operational knowledge of the system for the test to be conducted. It is based on the required specification of product.
- Black box testing is mostly concerned with observing the system behavior by providing the set of input that can be valid or invalid. System behavior is observed even in case of invalid set of input. From an end user is perspective a system valid input should not crash and be able to respond even in case of invalid input.
- There are various techniques of conducting black box testing. Few are listed below :
  - (a) Requirement based testing
  - (b) Positive and negative testing
  - (c) Graph based testing
  - (d) Boundary value analysis
  - (e) Decision table
  - (f) Equivalence partitioning
  - (g) Domain testing

☞ **Two most important black box testing methodologies**

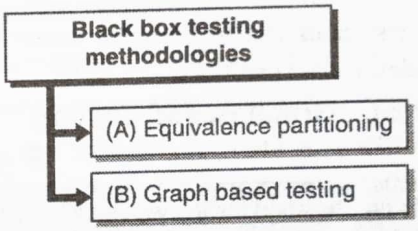


**Fig. C6.6 : Methodologies of black box testing**

### ➜ 6.4.2(A) Equivalence Partitioning

➜ **(May 2016)**

**Q. Explain the Equivalence class method of testing with one example. SPPU - May 2016, 6 Marks**

**Q. What is equivalence partitioning ? (4 Marks)**

- Equivalence partitioning is a black box software testing technique that divides the input data of system into partition of equivalent data and then test cases can be derived for partitions.
- Test cases are designed in such a way that each partition in covered at least once.
- The major advantage of this approach of testing is it reduces the time required in conducting the test as there are lesser number of test cases.
- Set of input data often considered as input values that can generate single expected output. Hence the single test case is designed for the entire partition.
- System behaves similarly for all inputs inside partitions.

☞ **Two major activities involved in equivalence partitioning**

(a) Identification of all partitions for complete set of input and output values.

(b) Selecting one member value from each partition for testing.

**Example**

- Take one example of system that accepts an integer value between 100 and 999.
- Valid equivalence class partition : 100 to 999 all are inclusive.
- Invalid equivalence class partition :
  - Less than 100
  - More than 999
  - Number with decimal
  - Alphabets
  - Non-numeric characters.
- Now, test cases can be designed based on their two equivalence class partitions.

## 6.4.2(B) Graph Based Testing

(Dec. 2016)

**Q.** Explain graph based testing with suitable example. **SPPU - Dec. 2016, 8 Marks**

- Graph based testing is also called state based testing and belongs to the category of black box testing.
- Graph based testing is particularly useful in the following scenarios :
  (i) The system is an automation of work flow where specific work flows are carried out depending on current state and input variables.
  (ii) System can be represented in the form of data flow model, where data flows from one state to another.
  (iii) System is a language processor like compiler and interpreter and can be represented in the form of state machine.
- Graph based testing is useful when a system transaction can be depicted as state transitions.
- Take an example of admission process in university. A typical admission processing system can be visualized as being made up of the following steps.
  1. Student fills up admission applications form online.
  2. Student appears in the admission test.
  3. Based on the examination performance
     (i) Result declared as selected
     (ii) Result declared as rejected.
  4. Student report to CAP centre for admission.

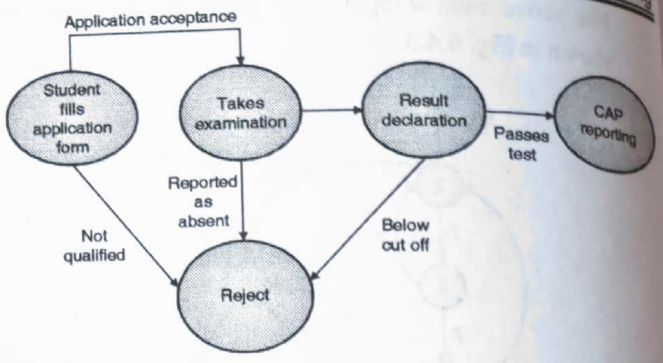


**Fig 6.4.4 : Example of possible state graph**

- For each of state in Fig. 6.4.4, the condition for state transition can be listed and next state can be identified. State graph testing verifies the conditions and state transitions.

## 6.4.3 Difference between White Box Testing and Black Box Testing

(May 2017)

**Q.** Differentiates between Black box testing and white box testing. **SPPU - May 2017, 8 Marks**

- Table 6.4.1 lists the major difference between white box and black box testing.

**Table 6.4.1**

| Sr. No. | Parameters | Black box testing | White box testing |
|---|---|---|---|
| 1. | Competency | Knowledge of internal structure of system is not essential to carry out the testing. | Knowledge of programming design and internal architecture must be known to the tester. |
| 2. | Testing Authority | Mostly independent test engineers are responsible to carry out the black box testing. | Software developers are responsible for doing such type of testing. |
| 3. | Objective | Tester mainly focuses on functionality of the system. | Developer focuses on program control flow, code logic, and structure of the system. |

| Sr. No. | Parameters | Black box testing | White box testing |
|---|---|---|---|
| 4. | Orientation | This type of testing focuses on what system is performing. | This type is testing focuses on how system is performing. |
| 5. | Type | This is functional test. | This is structural test. |

## 6.5 Phases of Testing

→ (May 2016)

**Q.** State the different phases of testing.
SPPU - May 2016, 1 Mark

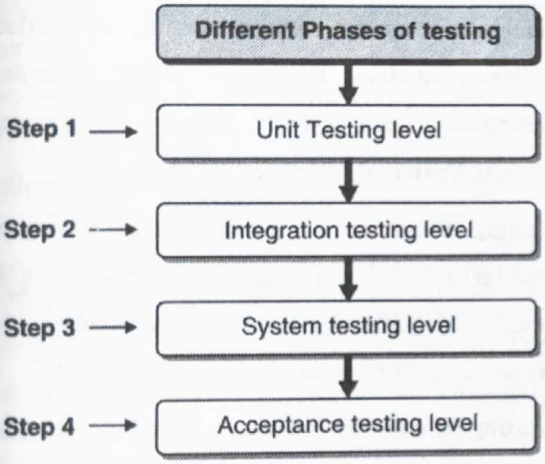


**Fig. 6.5.1 : Phases of testing**

**Syllabus Topic : Unit Testing**

### 6.5.1 Unit Testing

→ ( May 2016)

**Q.** Explain unit testing phase of testing.
SPPU - May 2016, 2 Marks

- Unit testing is software testing method where individual units of source code are tested after their completion.
- Unit is the smallest testable part of any system. It can be procedure, individual function or an entire module. It can be an entire class, interface or a single method.
- Unit tests are mostly performed by the developer who has created the unit of program.
- Unit test also includes test of control data for the units, usage procedures, operation procedures etc. These factors are tested to determine whether they are fit to use or not.
- Unit is the basis for component testing. Unit test is often performed as part of white box testing.
- Unit testing is the lowest level of testing in test hierarchy. In V model, unit test is designed by developers side by side of program implementation. After completion of code, unit test is executed.
- Units can be tested in isolation and independently. To assist such kind of unit test, special software components are used to drive the units of program. Stub programs, mock objects, fakes are example of such substitutes.
- A generalized flow of unit test is shown in Fig. 6.5.2. The test is executed with the source code and test cases. After test execution the result is reviewed by developer and test engineers, if not found satisfactory, piece of code is subjected for resubmission to unit test.

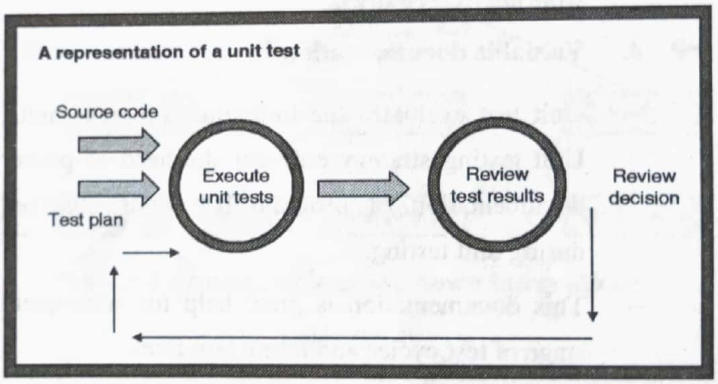


**Fig. 6.5.2 : A generalized flow of unit test**

### 6.5.1(A) Benefits of Unit Tests

**Q.** List and explain the benefits of unit test. **(6 Marks)**

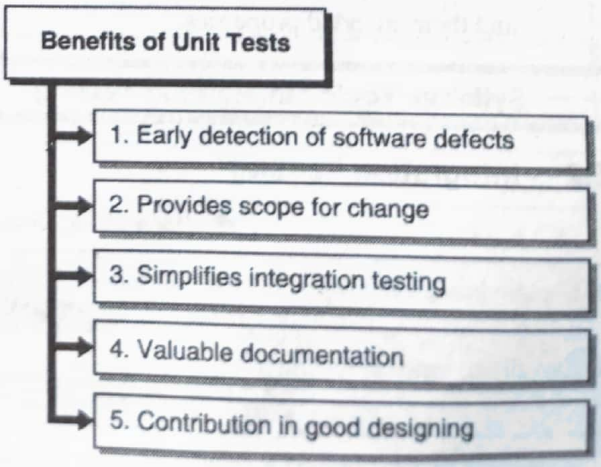


**Fig. C6.7 : Benefits of unit test**

➜ **1. Early detection of software defects**

- Unit test is performed on programs as soon as they developed. It does not wait for components integration.
- Often unit test cases are designed before the programming starts for particular module.
- Often the code is only considered as completed when it passes the unit test cases.

➜ **2. Provides scope for change**

Programmers can make changes in the library and code to make sure that their code is working properly or the patches applied are working properly at early stage.

➜ **3. Simplifies integration testing**

- Starting from unit testing is the bottom up approach of testing.
- So at the stage of integration testing, it has already been ensured that all the program units are working expectedly.

➜ **4. Valuable documentation**

- Unit test evaluates the individual program units. Unit testing strategy enforces the need of proper documentation of program behaviour observed during unit testing.
- This documentation is great help for subsequent stage of test cycles and future bug fixes.

➜ **5. Contribution in good designing**

- Unit test cases test design if written at early phase of software development life cycle, contributes in good designing.
- Unit test helps in specifying the object behaviour and their intended properties.

**Syllabus Topic : Integration Testing**

## 6.5.2 Integration Testing

➜ **(May 2016, Dec. 2016)**

**Q.** Explain integration testing phase of testing.
SPPU - May 2016, 2 Marks

**Q.** Explain integration testing .
SPPU - Dec. 2016, 6 Marks

- A system is implemented as components cooperating with each other. These components comprise of hardware and software. They are separately developed and later integrated with each other.
- The process of integrating the components and implementing the interaction between them is called **Integration**. The verification and testing of integration is called **integration testing.**
- Integration testing implies both, the type of testing and the phase of testing.
- Integration testing starts as soon as two or more components are available for integration and finally ends when the entire components are integrated and get tested.
- Integration testing requires additional information for performing the test. This information is about the communication and interaction of modules.
- Integration testing includes testing of interfaces for different modules that need to be integrated. The interfaces can be internal interfaces or exported/external interfaces.
- Testing of internal interfaces includes testing of those interfaces which are internal to the product. Internal interfaces are used for communication between modules within the product.
- Testing of external interfaces includes testing of those interfaces which are exposed to the clients or external systems.
- Interfaces are mostly Application Programming Interfaces (APIs) that can be tested along with its callers.
- Sometimes not all the interfaces are available for testing. In that case light weighted stubs are used to simulate the interfaces which are under development.
- Integration testing requires information of high level architectural design of the system. But all the interactions between the modules are not documented. Some interfaces are documented and some are not.
- The documented interface is called *explicit interfaces.* The interfaces which are known to the developers and test engineers but are not documented are called *implicit interfaces.*

- Integration testing requires testing of both of type interfaces : implicit and explicit.
- Integration testing involves several modules and their interfaces. One example is illustrated in Fig. 6.5.3.

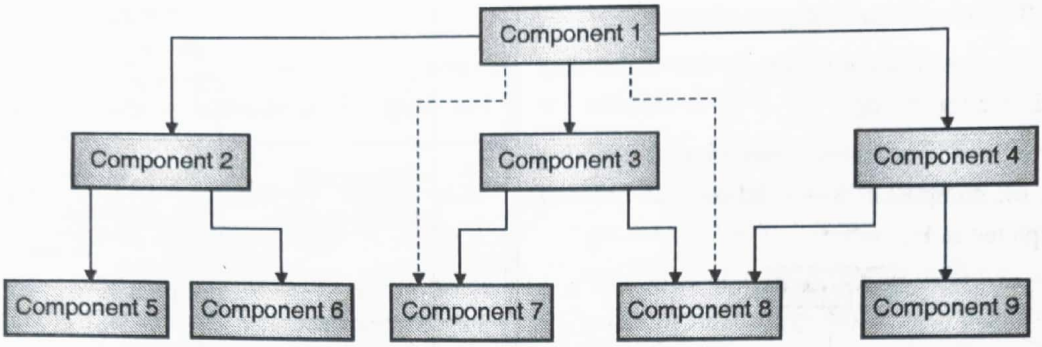


**Fig. 6.5.3 : Modules and interfaces to be tested**

- There are 9 components and 11 interfaces. Rectangles depicts the component and arrows depicts the interfaces.
- Two interfaces are explicit which are denoted by dotted arrow lines.

### Methods of integration testing

→ **(May 2016)**

**Q. What are different types of integration testing?** SPPU - May 2016, 2 Marks

- The order of integration testing can be defined by four methods.

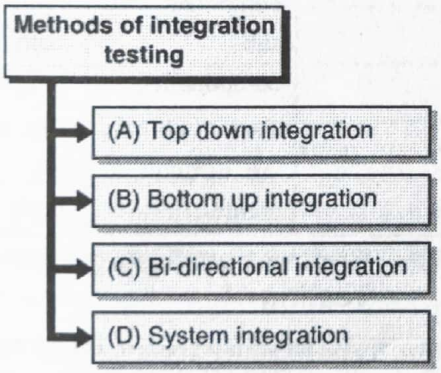


**Fig. C6.8 : Methods of integration testing**

## 6.5.2(A) Top Down Integration

- This method of testing specifies the testing of top most interface first, other component interfaces are tested in top to bottom order.
- Take an example where components becomes available for testing one by one.
- Suppose first component 1 and component 2 is available. So interfaces between component 1 and 2 is tested. One scenario is depicted in Fig. 6.5.4.
- A top to bottom approach of integration testing covers component to component top to bottom as module becomes available.
- Table 6.5.1 shows the order of module tested as they become available.

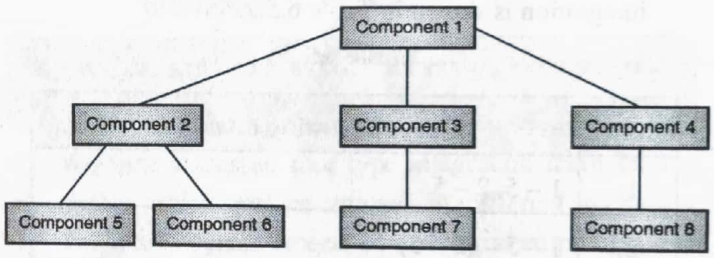


**Fig. 6.5.4 : An example of top down integrations**

**Table 6.5.1**

| Steps | Order of testing interfaces |
|---|---|
| 1. | 1 – 2 |
| 2. | 1 – 3 |
| 3. | 1 – 4 |
| 4. | 1 – 2 – 5 |
| 5. | 1 – 2 – 5 – (2 – 6) |
| 6. | 1 – 3 – 7 |
| 7. | 1 – 4 – 8 |
| 8. | 1 – 2 – 5 – (2 – 6) – (1 – 3 – 7) – (1 – 4 – 8) |

- Step 8 is the stage where all the modules become available for testing and integration, testing is performed for all interfaces.

## 6.5.2(B) Bottom-Up Integration

- Sometimes new modules or products become available in reverse order. In this case testing starts from bottom.

- Bottom-up approach is reverse of top down approach. one possible scenario is depicted in Fig. 6.5.5. Arrows pointing down specifies flow of logic or control. Arrows pointing up specifies integration paths.
- Components are assigned numbers in the order they become available for testing.
- Testing starts from component 1 and goes up covering all modules till component 8 is reached. The order of testing is depicted in Fig. 6.5.5.

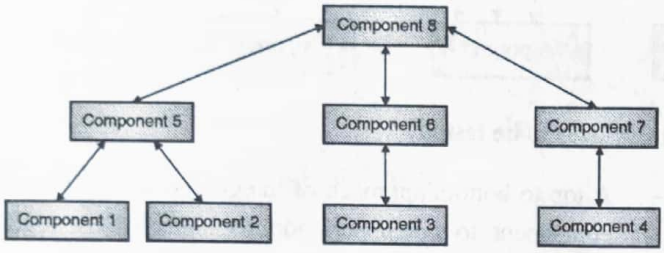


**Fig. 6.5.5 : An example of bottom-up integration**

- The steps and order of testing in bottom up approach of integration is shown in Table 6.5.2.

**Table 6.5.2**

| Steps | Order of testing interfaces |
|---|---|
| 1. | 1 – 5, 2 – 5 |
| 2. | 1 – 5 – (2 – 5) |
| 3. | 3 – 6 |
| 4. | 4 – 7 |
| 5. | 1 – 5 – (2 – 5) – 8 |
| 6. | 3 – 6 – 8 |
| 7. | 4 – 7 – 8 |
| 8. | (1 – 5 – (2 – 5) – 8) – (3 – 6 – 8) – (4 – 7 – 8) |

- Observing closely the approach of bottom-up testing, it can be stated that the approach is well suited for water fall, V - model or modified V model that uses bottom up approach of testing.

☞ **Difference between Top Down Integration and Bottom Up Integration**

**Q.** Compare top-down and bottom up integration. **(5 Marks)**

| Sr. No. | Parameters | Top down Integration | Bottom up Integration |
|---|---|---|---|
| 1 | Commencement | Testing can be started at early stage of software development life cycle. | Testing starts later stage of system implementation when starts becoming available. |
| 2 | Supporting Components | Stubs or fakes are required in top down approach | Driver modules are needed in bottom up approach |
| 3. | Design Verification | Design decisions can be verified in the testing at early stage | Later stage of testing leaves very less scope for system design verification |
| 4 | Implementation | Fast implementation is possible with early phases of testing. | Running system is available only after complete integration |
| 5 | Orientation | Top down testing starts from main component to sub components. Test conditions are difficult to create. | Bottom up testing starts from sub component to main component. Test conditions are relatively easily created. |

## ➜ 6.5.3(C) Bi-directional Integration Testing

- Top down approach of integration testing and bottom-up approach both are combined to implement bidirectional integration testing.

Consider the arrangement of components shown in Fig. 6.5.6.

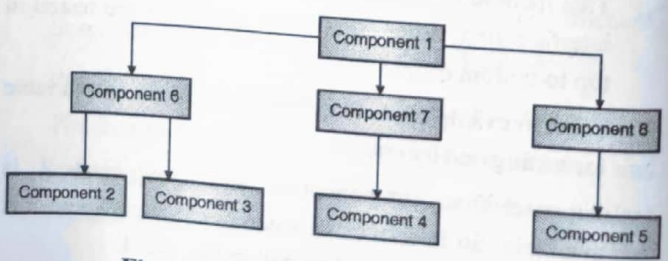


**Fig. 6.5.6 : Bidirectional Integration**

- In this arrangement, bidirectional testing is initially performed with the help of stubs and drivers. Only few components are available initially which are tested separately.
- In Fig. 6.5.6, components 1, 2, 3, 4 and 5 are supposed to be tested individually with the help of drivers and stubs.
- Upstream connectivity is provided by drivers while downstream connectivity is provided by stubs.
- Drivers and stubs are used to simulate the operations of the components which are still not available and are under development.
- Once the functionality gets tested, the drivers and stubs remain of no use.
- Once the intermediate components become available they are combined with already tested module and tested again using bidirectional approach. The approach is sometimes called "Sandwich integration".
- Steps of bidirectional testing for the arrangement of components shown in Fig. 6.5.6 are listed in Table 6.5.3.

**Table 6.5.3**

| Steps | Order of testing interfaces |
|---|---|
| 1. | 6 – 2 – 3 |
| 2. | 7 – 4 |
| 3. | 8 – 5 |
| 4. | (1 – 6 – 2 – 3) – (1 – 7 – 4) – (1 – 8 – 5) |

- This kind of bidirectional testing is extremely useful when existing system is getting enhanced or new functionality is getting added in the existing system.
- One example is when a two tier system architecture is getting converted into three tier architecture and several new modules are added at middle layer to form the new tier.

### 6.5.2(D) System Integration

- When all the components are integrated and tested together as a single unit, the testing is called system integration testing.
- Integration testing is classified into two parts :
  - Components integration
  - System integration
- This kind of testing test the entire system as an integrated unit, instead of testing each component individually step by step.
- This kind of testing is also called "big bang testing" and the kind of integration is called "big bang integration".
- Big bang integration is well suited for system development where interfaces have very less chances to get detected with defects.
- This kind of testing is well suited in a scenario when most of the components and interfaces are already available for testing. When there is very less chances of system enhancements with new modules.
- This type of testing provides simplicity but has disadvantages too. It is sometimes hard to detect the point of failure in case of any defect in entire integrated system.

### 6.5.2(E) Application of Different Integration Methods

**Q. State the application of different integration methods. (4 Marks)**

- We have discussed four type integration methods and testing which can be applied in different scenarios. There are several aspects of development process and nature of project that drives the application of integration testing.
- Few factors that should be considered while choosing the type of integration method is listed in Table 6.5.4.

**Table 6.5.4**

| Integration Method | Scenario |
|---|---|
| Top down integration | When design and requirements are clear and concrete. |
| Bottom up Integration | When requirements cannot be freezed initially and can change dynamically.<br>Where design or architecture can also be changed at later stage of development. |
| Bidirectional Integration | When new enhancements have to be made in system. Design is stable but architecture is going to be changed. |
| System Integration or Big Bang | There are very less chances of change in architecture. Less chances of defects to be detected in interfaces. |

### 6.5.3 System Testing

**Q. Explain system testing. (4 Marks)**

- When the testing is performed on complete integrated system capable of performing all the intended operation, the testing is called system testing.
- Normally, it is performed after unit and integration testing. There is very thin line of separation between system integration testing and system testing.
- System testing checks for all intended features of the system. The system is evaluated against all the requirement specifications.
- In case system is composed of hardware and software, then system testing include the entire system as a whole.
- System testing is performed to test both functional and non functional requirements of the system.
- Functional requirements include the actual real life requirement of customers. Non functional requirement includes several quality aspects of the system.

Non functional requirements categorize the system testing in several distinct classes. They are listed in Fig. C6.9.

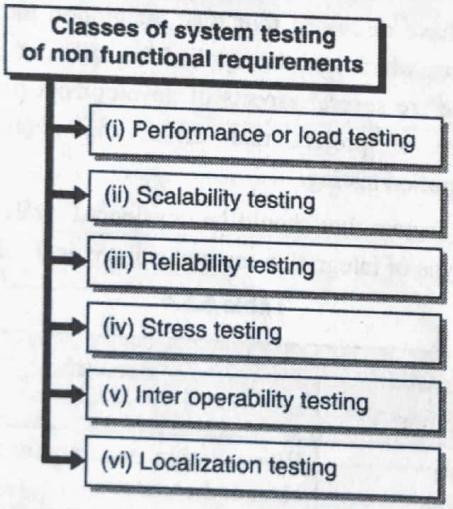


**Fig. C6.9 : Classes of system testing of non functional requirements**

➜ **(i) Performance or load testing**

Performance testing evaluates the system on the basis time taken or response time of the system that it takes to perform the required tasks.

➜ **(ii) Scalability testing**

Scalability testing is concerned with testing of maximum capability of system. It requires significant amount of resources to conduct scalability testing.

➜ **(iii) Reliability testing**

Reliability testing evaluates the ability of system to perform its required functionality repeatedly without fail.

➜ **(iv) Stress testing**

Stress testing evaluates the system by overloading it beyond the limits of specified requirements. It ensures system should not go down or fail in these case and should able to react positively.

➜ **(v) Inter operability testing**

Interoperability testing ensures that the system can communicate and co-operate with other system that is supposed to work together. It ensures two or more cooperative system or products can exchange the required information and use the information without any difficulty.

➜ **(vi) Localization testing**

The testing is done to ensure the language compatibilities of the system and to ensure product works effective with different intended localities.

### 6.5.3(A) Need of System Testing

➜ **(May 2016)**

**Q. What is need of system testing? SPPU - May 2016, 4 Marks**

Primary need of system testing is listed below :

(i) It facilitate independent view of system and its testing.

(ii) It captures customer's perspective in the system for the first time.

(iii) The entire system has to be test for its overall functional and nonfunctional requirement which is not in captured during unit, component or integration testing.

(iv) It develops the confidence for all stakeholders in the product.

(v) It reduces the risk in product and provides a smooth transition towards product release.

(vi) Ensures that product is now ready for acceptable testing.

### 6.5.3(B) Difference between Functional and Non Functional Testing

**Q. Differentiate between functional and non-functional testing. (6 Marks)**

- System functionality and features are tested in functional testing. Product's quality factors are tested in non-functional testing.
- Functional testing is concerned with verification of the tasks that system is supposed to perform. It has simple methods for test case execution and does not depend upon environment of the system.
- Non functional testing is performed to verify the quality of system. There quality includes reliability scalability, etc. It requires larger amount of resources to carry out the test and some time depends on system's environment also.
- Functional testing requires deep knowledge of product's behavior and expectations. Non functional testing requires knowledge of system architecture and environment.
- Functional testing verifies the requirements in terms of 'met or not met'. Non functional testing checks for result documented in 'qualitative and quantifiable' terms.
- Smaller set of selective data can be used for functional testing but non functional testing needs large amount of data with all variations.
- The summary of difference between functional and non functional testing is listed in Table 6.5.5.

**Table 6.5.5 : Difference between functional and non-functional testing**

| Sr. No. | Functional Testing | Non Functional Testing |
|---|---|---|
| 1. | It tests product behavior. | It tests product's quality and non function behavior. |
| 2. | It include products features and functionality. | It includes all quality factors. |
| 3. | Testing focuses on defect detection. | Testing focuses on qualification of product |
| 4. | Failure is detected due to defect in code (mostly) | Failure is detected due to defect in architecture, design or code. |
| 5. | Test can be repeated many times | Test in repeated in case of failures. |
| 6. | The phase of functional testing includes unit, component, integration and system testing. | The phase of non functional testing only includes system testing. |

## 6.5.3(C)Types and Techniques of Functional System Testing

**Q.** Briefly explain different types of functional system testing. **(5 Marks)**

There are several possible techniques to perform functional system testing some important techniques are listed in Fig. C6.10.

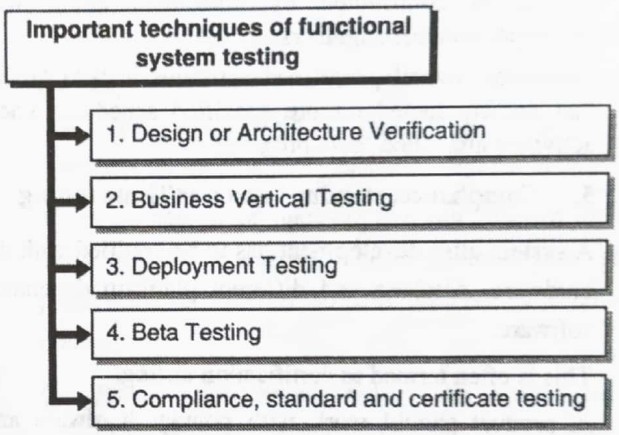


**Fig. C6.10 : Important techniques of functional system testing**

➜ **1. Design or Architecture Verification**

Test cases are developed with respect to the design and architecture of the system. Such test cases are also called product level test cases. It concentrates on behavior of complete product.

➜ **2. Business Vertical Testing**

- Business vertical testing is concerned with the product that automates different business activities of an organization like: Banking system, payroll, employee management system etc.
- There system automates the work flow of an organization and testing requires understanding of different business verticals.
- In such kind of testing business operations are verified. Testing procedures in such "system testing" are slightly modified to suit the business products requirement.

### 3. Deployment Testing

- Deployment testing ensures that customers deployment requirements are met. This testing includes testing of the product in user environment and platform.
- It is mostly conducted after the release of the product in customer's premises.

### 4. Beta Testing

- When there is high risk of product rejection by customers and continuous feedback is needed by customers in final stages of development, beta testing is conducted.
- A product under test is sent to the customers and feedback is continuously collected based on customers use and operation.
- Testing is contributed by customers along with developer and test engineers.
- There are several predefined activities in beta testing that are conducted on pre specified schedule. There activities are called 'beta program'.

### 5. Compliance, standard and certificate testing

- A system after development has to be certified with the hardware, database and different platform dependant software.
- This is often termed as certification testing.
- A product should work with popular hardware and software otherwise its user acceptance will decrease some popular compliance testing are listed below :
  (a) 508 accessibility guideline
  This is set of guidelines that make any software usable for physically challenged users.
  (b) SOX (Sarbanes - Oxley's Act)
  This act put the requirement of product audit in order to prevent financial fraud in the company.
  (c) OFAC and patrior act
- This is applied banking system transactions. The transaction should be audited to ensure misuse of large funds.

**Syllabus Topic : GUI Testing**

## 6.5.4 GUI testing

**(May 2016)**

Q. Explain GUI testing. **SPPU- May 2016, 4 Marks**

- GUI (Graphical User Interface) testing is the set of techniques to conduct the testing for product's graphical user interface.
- The set of techniques ensure that product's GUI meets all the required specification and does not contain any defect in its operation. It checks design of the application.
- Variety of test cases are designed to conduct the GUI testing that may differ from product to product.
- All test cases written for GUI testing together ensure the proper functionality of graphical user interface for the product. They also make sure that GUI conforms to its documented specifications.
- GUI testing also evaluates design elements like layouts, labels, text, font, color, captions, buttons, lists, icons, links etc.
- GUI testing can be manual or can be an automated process. It can be conducted by internal team in the organization or can be conducted by third party experts.
- End user can also be involved in GUI testing. Client's representative or business analyst can contribute in user interface evaluation process.
- GUI testing is time consuming. It is highly desired that while designing and implementing GUI elements, developer should write intended functionality of the GUI element so that tester will be aware of expected outcome.
- It gives extra clarity to tester on what exactly is expected from the GUI element.
- GUI testing is not just limited to testing of visual components of system. It checks for the related program functionality and behavior that triggers any GUI event or which is affected by some GUI event occurrences.
- For better understanding of scope and nature of GUI testing consider the following example checklist that an organization can prepare for its specific product :
  - Check that the intended functionality of system can be executed successfully by GUI.
  - Check all the GUI elements for different parameters like size, position, length, acceptance of alphanumeric characters etc.
  - Check all the error messages are getting displayed as specified in requirements.
  - Check images has good clarity and font is readable.
  - Check the different GUI elements are properly aligned.

- Check the GUI elements in different desktop environment or browsers.
- Check the position of GUI elements for different screen resolution.
- Check usability conditions. Verify all navigations.

### 6.5.4(A) Methods of GUI Testing

**Q. What are different methods of GUI testing? (4 Marks)**

In general, GUI testing can be performed using three approaches.

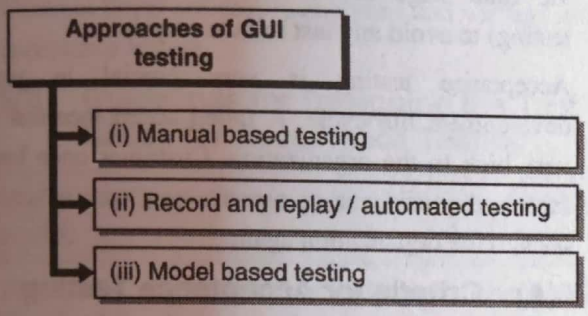


**Fig. C6.11 : Approaches of GUI testing**

➜ **(i) Manual based testing**

Test engineers and developers manually checks the graphical screens as per the requirements stated in requirement specification document.

➜ **(ii) Automated or record and replay testing**

- GUI testing is performed by automated tools. In first stage record of test steps are captured in GUI testing automation tool.
- Second stage is playback stage where recorded steps are executed and closely observed.

➜ **(iii) Model based testing**

- The GUI system behavior is captured in graphical model. Model captures the steps of GUI operation sequence. It can be modeled as state transition diagram.
- Model helps in designing and executing test cases more efficiently. Charts and decision tables are derived from model to formulate test cases.

### 6.5.4(B) Planning of GUI Testing

**Q. Explain the planning of GUI testing. (4 Marks)**

- GUI testing needs a well planned strategy to carry out test case design and execution.
- Test suite is generated initially in same way as it has rather been generated for CLI (Command Line Interface).
- Later on test suite is adapted from CLI to GUI equivalent and additional GUI parameters are added for verification.
- Planning often targets to adapt model based technique for conducting GUI testing. It involves following four stages :
  - (i) Initial state
  - (ii) A goal state
  - (iii) Set of operators
  - (iv) Set of objects to be operated.
- Planning of GUI testing involves identification of all the above listed elements or parameters. It specifies the path from initial state to goal state using specific operators on specific objects.
- There are automated planners that can generate the test cases for GUI testing. These planners are often preferred over manual process because of following reasons :
  - (i) Planners are always valid with defined set of operators and objects planner is capable of generating either a valid test case or no test cases at all.
  - (ii) The order of GUI testing is effectively managed by planner. Sometimes GUI test cases are required to be followed in very specific order. Manual ordering of these activities are erroneous.
  - (iii) A planning must be goal oriented. Tester can specify and select what is most important for product and organization. Then test cases and plan will get generated accordingly.

### 6.5.4(C) Web Based Application Testing

➜ **(May 2016)**

**Q. Explain the web based testing.**
**SPPU - May 2016, 4 Marks**

Web based testing is specific methodology of testing used for verifying and validating the applications hosted on web.

**Primary components of web based application testing**

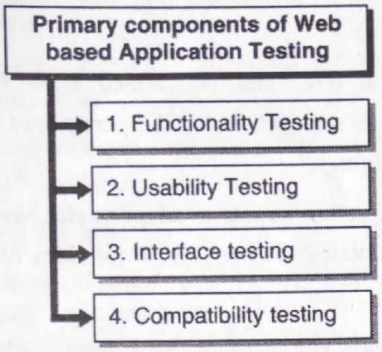


**Fig. C6.12 : Components of web based application testing**

➜ **1. Functionality Testing**

This includes validating all fields of application, its proper functionality and workflow.

➜ **2. Usability Testing**

This include testing from users perspective and intuition. It also include content checking.

➜ **3. Interface testing**

This includes testing of the interfacing of web application with other systems. It includes test of data flow to other systems

➜ **4. Compatibility testing**

It includes testing of compatibility of software with different devices, operating system, browsers etc.

## Syllabus Topic : User Acceptance Validation Testing

### 6.5.5 User Acceptance Validation Testing

**Q. What is acceptance testing? (4 Marks)**

- Once the system testing is completed, another phase of testing starts called the acceptance testing or validation testing. This is the highest level of testing and found at top of the testing hierarchy.
- Acceptance testing is normally done by customer, customer's representative or business analysts.
- Test cases are often designed directly by customer or by the client's representative. Sometimes it is designed by test engineers with consultation of customers. These test cases are executed to check quality and acceptance of product.
- A major point in the life cycle of a product is when it is purchased by the customer. Customers often run the user acceptability test cases before purchasing the product.
- Acceptance test cases are not concentrated on finding the defects in the system as it is already covered in system testing. There design is bases on practical use case scenarios.
- Often test cases are designed jointly by development organization and customers. Development organization often prefers to run those test cases in advance during the final stage of development (along with system testing) to avoid any last minute surprises.
- Acceptance testing is very crucial in product development life cycle. A failed acceptance test costs very high to the organization. Customer once lost the faith and confidence in organization may never come back to the organization again.

### 6.5.5(A) Criteria for Acceptance Testing

**Q. Briefly explain different criteria for acceptance testing. (5 Marks)**

Following are the most common classes of acceptance where acceptance criteria are applied.

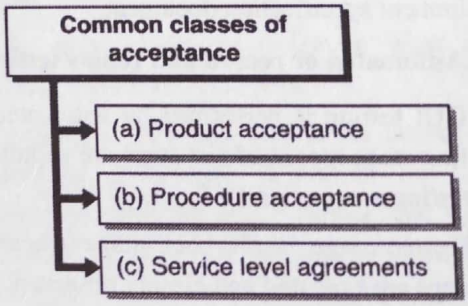


**Fig. C6.13 : Common classes of acceptance**

➜ **(a) Product acceptance**

At the first stage of product development i.e. requirement specification, each requirement is linked with acceptance. Example of procedure based acceptance criteria. This may include test case from different test phases. Most of the non-functional requirement forms the basis of product acceptance criteria. Testing for compliance, standards etc. can also be part of acceptance criteria.

➜ **(b) Procedure acceptance**

The procedures which are followed for product delivery forms the basis of acceptance testing for procedure acceptance. Example of procedure based acceptance criteria are listed below:

(i) Product should be delivered with all associated documentation.

(ii) Source code has to be provided along with the executable files.

(iii) A team development organization will conduct knowledge transfer sessions with client's employee.

➜ **(c) Service level agreements**

Service level agreements are often taken as part of acceptance criteria. This is the set of contracts signed mutually by client and development organization. Contract items are deeply analyzed and verified during acceptance testing.

## 6.5.5(B) Guidelines for Designing the Test Cases for Acceptance Testing

**Q.** Briefly explain different guidelines for acceptance testing. **(5 Marks)**

Fig. C6.14 shows the few guidelines which can be considered while designing the test cases for acceptance testing.

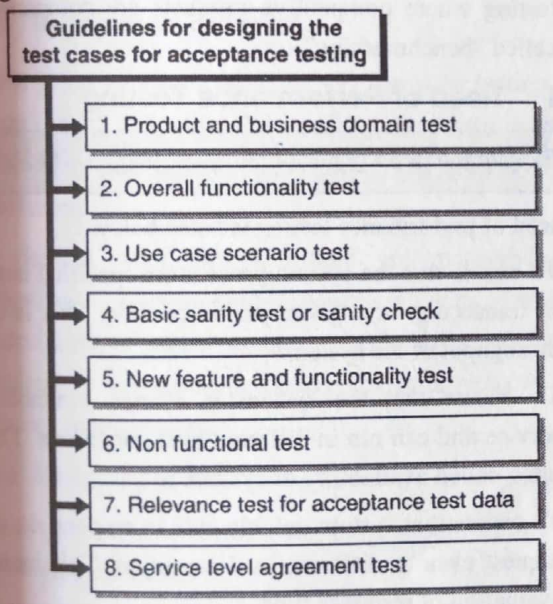


**Fig. C6.14 : Guidelines for designing test cases for acceptance testing**

➜ **1. Product and business domain test**

Acceptance test is primarily concerned with business acceptance and hence requires test cases based on business activities of customer. It focuses on business scenarios.

➜ **2. Overall functionality**

This is end to end functionality test of the product. A product as a whole is tested for acceptance testing. It tests all the business transaction of the system and their results.

➜ **3. Use case scenario test**

It includes all the possible use case verification in the product. Use case document is referred for acceptance test.

➜ **4. Sanity check**

Sanity test or sanity check includes the testing of claimed behavior of the system. It is a quick test to evaluate the claim about the product. It ensures system is performing all its intended operation. It highly recommended to perform sanity check when product undergoes with major changes.

➜ **5. New features and functionality**

This tests is concentrated on testing of newly added features and functionality along with its sanity check.

➜ **6. Non-functional test**

Non functional testing for quality of product is also included in acceptance testing. Test cases are designed to include features and functionality test along with the quality of those features.

➜ **7. Relevance of data used in acceptance testing**

It is important to verity the data and its relevance in the test to be conducted. It is highly recommended to make use of actual real life data of customers for test case execution.

➜ **8. Service level agreement test**

Acceptance test criteria and test cases should be designed to include product compliance, standard and certificates.

**Syllabus Topic : Scenario Testing**

## 6.6 Scenario Testing

- Scenario testing is a testing activity where end to end product testing is carried out with the help of hypothetical scenarios. These are operational scenarios and quite useful as it directly related real life product usage.
- The scenarios are formulated in such a way that it will be easy to evaluate and remain credible.
- A scenario can have many test steps or test cases. It is important to understand that test scenarios are different from test cases.

- Scenario testing helps in determining that end to end functionality of the software system is working as expected.
- Scenario testing helps test engineers and developers to understand how system will behave in the hand of end users.
- Critical business level defect can be recognized in the scenario testing as it is based on small business stories.
- Scenarios can based on system scenarios, use case scenarios or the role based scenarios.
- Following are the few strategies to create good scenarios to implement scenarios testing.
  - Find out all potential users of the system, their work flow or activity flow with the system, their actions and objectives.
  - Perform evaluation of user's mindset of system interaction. Find the systems usage and behavior from hacker's perspective. Explore all system abuses.
  - Formulate a framework to track all system events and responses.
  - Explore more similar systems and other scenarios system reports. Compare it with newly formed system.

**Syllabus Topic : Performance Testing**

## 6.7 Performance Testing

**Q. What is performance testing ? (2 Marks)**

- Performance is one of the important factors based on which a system is evaluated by developers customer and general users.
- The business of a software development organization is highly influenced by the performance of their product. A high performing system provides scope for expanding the business for an organization.
- Reliability, scalability, speed, throughput, latency etc are often considered as major factors affecting the overall performance of the system.
- We have discussed in earlier sections the non functional requirements of system that affect the performance. We will briefly discuss few more factors like throughput, response time, latency that plays major role in system performance.

**Throughput**

- The system capability of handing multiple request or transaction in a given period of time is determined by a factor called "through put". It can vary as different load is subjected to the system.

**Response time**

- Response time is considered as the time duration between request and the first response from the system. General users have a habit of evaluating the system based on response time.

**Latency**

- Latency is the delay caused by the factors external to the system. The delay often caused by operating system, supporting platform or external environment. These delays are calculated separately.

**Performance**

- Testing include evaluation and verification of all these primary components. Often competitors product is evaluated for performance testing. Such performance testing where competitive products are compared are called "benchmarking".

### 6.7.1 Need of Performance Testing

**Q. Explain the need of performance testing. (4 Marks)**

The need of performance testing is listed below :

(i) To ensure that the system process the specified number of transaction at a given period of time. This is often throughput of the system.

(ii) To ensure that the system is always available for service and can run in different load conditions. This is often called availability of system.

(iii) To ensure that system quickly able to respond on users request even at different load condition. This includes evaluation of response time.

(iv) To ensure that the product is comparable to other competitive products and should be better than others.

(v) To ensure that the return of investment for resources being used is acceptable. Also to decide what resources are needed by system to increase the overall performance of the system.

### 6.7.2 Activities of Performance Testing

**Q. Explain different activities of performance testing. (4 Marks)**

Major activities and methodologies involved in performance testing is listed in Fig. C6.15.

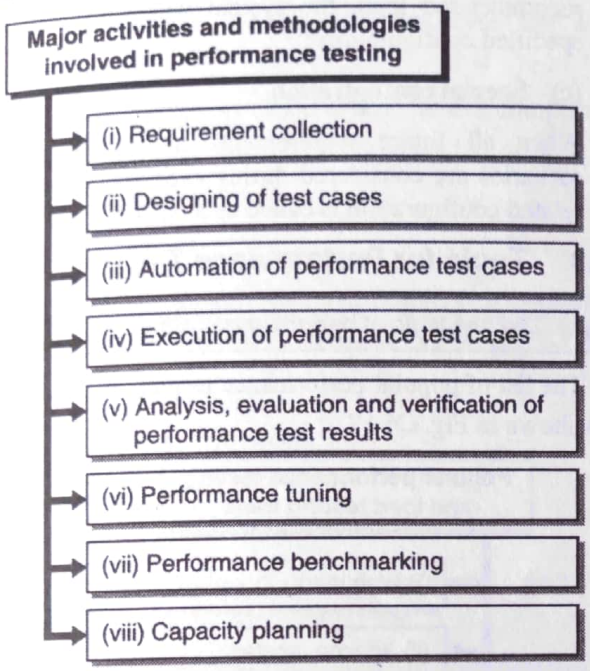


Fig. C6.15 : **Major activities and methodologies in involved in performance testing**

➜ **(i) Requirement collection**

First steps in the planning of performance testing are requirement collection. It is decided that based on what factors performance of the system is going to be evaluated.

It decides what factors should get measured and improved. Requirement collection also includes calculation of actual number of improvement that is desired or the actual percentage of improvement that desired in the system.

➜ **(ii) Designing of test cases**

Test cases for performance testing should define the following elements. :

(a) List of all operations to be tested

(b) Steps involved in executing the listed operations.

(c) List of external factor that affects the performance of system.

(d) Patterns in load variations.

(e) Supporting resources and their configuration.

(f) The expected result.

(g) Details of product to be compared for benchmarking.

➜ **(iii) Automation of performance test cases**

Automation of performance testing in highly desire because of following reasons :

(a) There is need of repetition of performance testing at regular intervals.

(b) Sometimes performance testing such huge amount of data that manual testing is not possible.

(c) Accuracy in result is highly desired.

(d) There are several factors affecting the performance testing. Sometimes the combination of those factors should be taken into account. Such permutation combination is not possible is performance testing if done manually.

(e) Analysis of result is also a cumbersome task which is performed by automation tools.

➜ **(iv) Execution of performance test cases**

Execution of performance test case requires data to be collected which should be relevant to the performance. Data related to the following element need to be collected :

(a) Start time and completion time of test case execution.

(b) Log and audit files of supporting platform.

(c) Statistics related to CPU – disk and memory utilization.

(d) Configuration data of application and supporting platform.

(e) Response time, through put and latency specified in test cases.

➜ **(v) Analysis of performance test results**

Analysis of performance test result can take multiple forms. The techniques of analysis can vary organization to organization. It requires expert analysts with statistical backgournd to evaluate the result.

Analysis can be performed with respect to following factors :

(a) Whether performance varies for different load and different system configuration.

(b) Whether the performance is constant when the same test is performed multiple times.

(c) What are the factors and parameters which has major impact on system performance.

(d) What is the maximum load limit which is acceptable.

(e) What should be optimal values for throughput, latency and response time.

(f) What requirement of performance are met what remains unfulfilled.

➜ **(vi) Performance tuning**

A successful analysis of performance test result enables the organization to tune the performance by adjusting different parameters and configuration of system. This is called performance tuning. There are two types of performance tuning.

(i) Tuning of application (product) parameters.

(ii) Tuning of external environment parameters (Like O.S, hardware).

➜ **(vii) Performance benchmarking**

Performance benchmarking includes the comparision of product with competitive products. Following are the steps of conducting performance benchmarking.

(a) Selecting the eligible competitive products for benchmarking.

(b) Identifying the factors based on which comparison has to be made.

(c) Comparing the performance of selected products for identified factors.

(d) Tuning the necessary parameters for best comparison.

(e) Documenting, publishing and reporting the result of performance bench marking.

➜ **(viii) Capacity planning**

Capacity planning is the process of identifying and recommending right configuration to the customers. It includes three types of configurations.

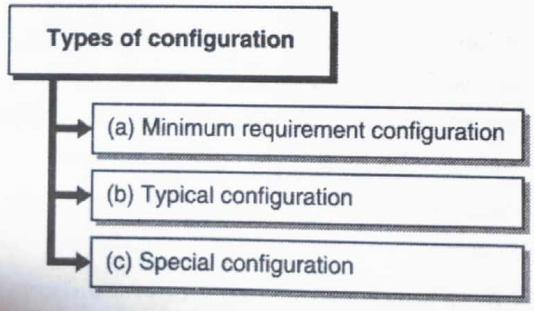


**Fig. C6.16 : Types of configuration**

➜ **(a) Minimum requirement configuration**

It specifies that any configuration less than the specified configuration may result in unexpected behavior of system.

➜ **(b) Typical configuration**

Typical configuration specifies that under recommended load, the system will work fine with specified configuration.

➜ **(c) Special configuration**

When all future requirements and worst possible scenarios are considered during capacity planning, the related configuration is called special configuration.

### 6.7.3 Tools for Performance Testing

**Q. List few tools of performance testing. (2 Marks)**

The list of popular performance testing and load testing tools shown in Fig. C6.17.

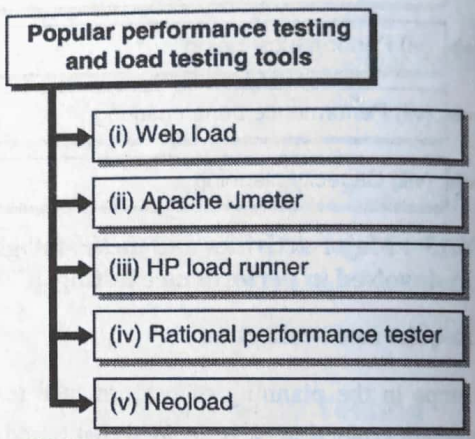


**Fig. C6.17 : Popular performance testing and load testing tools**

➜ **(i) Web load**

Web load is performance load testing tool for web based applications.

➜ **(ii) Apache Jmeter**

This is an open source load testing tools for java applications. It has support for functional test plan.

➜ **(iii) HP load runner**

This is another load testing tool by Hewlett-packard.

➜ **(iv) Rational performance tester**

This is an IBM tool for automation of performance testing.

➜ **(v) Neoload**

This is a tool for measuring and analyzing performance of web sites.

**Syllabus Topic : Test Cases and Test Plan**

## 6.8 Test Cases and Test Plan

**Q. Explain test cases and test plan? (6 Marks)**

**Test Case**

- The singular unit of test scenario is a test case. There can be positive and negative test case for a test scenario.
- Test case is often specified as a document comprises of the following components:
  - Test data
  - Procedures
  - Inputs
  - Scenarios
  - Descriptions
  - Testing environment
  - Expected results
  - Actual results
- Test case is designed with the primary objective of finding bugs in the software system.
- Test cases are also designed to demonstrate the execution of application when it produces the correct result.
- Test cases helps in a real world application demonstration and gives a direct idea of how the system can used in all possible ways by the user.

**Test Plan**

- Test plan has broader perspective and covers the entire strategy of system testing along with documented plan of testing. The test plan comprises of the following:
  - Scope of the project
  - Objectives
  - Target Customers and users
  - Assumptions
  - Test start date
  - Test end date
  - Roles and responsibilities of testing
  - All testing resources
  - Testing environment
  - Deliverables
  - Major risks during the test
  - Risk handling mechanism
  - Defect reporting and mitigation
- Test plan exists in the form of formal documentation that can be referred again and again during the testing and test case formulation. The test plan is prepared by test engineers and other stake holders if needed.
- Test plan should be clear, concise and readable. It should not be complicated.

**Syllabus Topic : Case studies expected for developing usability test plans and test cases**

## 6.9 A Case Study : Developing Usability Test Plans and Test Cases

**Q. With the help of an example, formulate test plan and test cases for real life application. (5 Marks)**

- Consider the following system and let us try to develop a test plan and test cases for the system.
- There is a medical centre in college campus serving as small hospital for the college employees and students. A system needs to be developed that automates the daily routine tasks of medical centre. We can call it as Medical Centre System or Medical Centre Application. This should be a web based application so that patients can access it remotely. Patient has to register the personal information for the first time and will get an access card which can be used for all subsequent interaction with the system. A unique patient Id will be generated along with access card.
- Patient can also visit the hospital, contact the help desk and get the personal information registered in the system. Patient can fix an appointment, check the schedule, access the lab report using the online portal or can access the medical centre application system from inside the hospital.
- Doctor consultations can be queued and managed by the system using tokens. Doctor's assistant can type the prescription and save in system. Doctor can access the patient's history.
- Pathologist and lab assistants can access the prescription and save the report. Patients can upload the previous reports and download the report from the system.

**Sample test plan**

- A sample test plan can be formulated shown in Table 6.9.1. Note that a formal test plan will exist in the form of complete planning report. For the sake of simplicity we are taking the summary (sum-up) of report and representing it in tabular format.

**Table 6.9.1**

| Sr. No. | Planning Elements | Description |
|---|---|---|
| 1 | Objective | Specifying overall objective of the testing. It exactly specifies what is expected from this test plan. |
| 2 | Scope | It exactly specifies what part of the medical center application is covered in testing. In general, the entire system is intended to be covered in testing. |
| 3 | Assumptions | Assumptions if any is specified in this planning element. An interface for payment of fees that will connect the system to an external entity can be specified here. Such interface require specific assumptions. |
| 4 | Test Start Date | This has to be decides based on scope, feasibility, available testing resources and other deadlines. |
| 5 | Test End Date | This has to be decides based on scope, feasibility, available testing resources and other deadlines. |
| 6 | Roles and responsibilities | Roles and responsibilities of the available testing resources are specified for carrying different testing. |
| 7 | Testing resources | This includes the human resource and skilled employees involved in testing. It also include other indirect audiences participating in the testing. |
| 8 | Testing Environment | The environment of testing is specified. It should be exact replica of production environment. Even it is possible for medical center application that the system is deployed in medical center premises for testing. |
| 9 | Deliverables | Test ddeliverables are the collection of all the documents, tools and other components that has to be developed and maintained in support and help of the testing. |
| 10 | Risk | This include all the risk associated with testing. This clearly specifies which part of the system is hard to test and unpredictable. |

☞ **Test case**

– For the above stated medical center application there can be many **test cases** formulated based on the use cases. We will take one test case for demonstration. Note that test cases can be formulated and represented in many ways. One simple form is shown here. Also note that the Table 6.9.2 shows test case is derived from the system use case.

**Table 6.9.2**

| Step | Procedure/Input | Expected Result |
|---|---|---|
| The patient select an option to submit an appointment request | Select submit appointment request application | System shows a blank form for appointment request. |
| The system prompts for the information to be filled in appointment form | None | |
| The patient enters the information | Enters information: Ravi Ranjan, Dr Neeraj Gupta, Ophthalmology, | System shows message "your appointment request is under |

| Step | Procedure/Input | Expected Result |
|---|---|---|
| | 12/10/2017, OPD, 10 AM | process" |
| The system validates the information is correct and requested doctor/slot is available | None | |
| The system displays the confirmation | Patient views the confirmation | System shows the conformation message with required details |

## 6.10 Exam Pack (University and Review Questions)

### Syllabus Topic : Introduction to Testing

Q. Explain the basic concept of software testing and its importance. *(Refer section 6.1)* (4 Marks)

Q. State and explain different software testing principles of testing. *(Refer section 6.1.1)* (6 Marks) **(Dec. 2016)**

Q. Explain different testing strategies. *(Refer section 6.1.2)* (6 Marks) **(May 2016)**

### Syllabus Topic : Error, Faults and Failure

Q. Write short note on error in software testing. *(Refer section 6.2.1)*(4 Marks)

Q. Define fault. *(Refer section 6.2.2)* (2 Marks) **(Dec. 2015)**

Q. Define failure. *(Refer section 6.2.3)* (2 Marks) **(Dec. 2015)**

### Syllabus Topic : Verification and Validation

Q. Explain verification and validation of software system. *(Refer section 6.3)* (6 Marks)

Q. Explain the difference between software verification and validation with an example. *(Refer section 6.3.1)* (6 Marks) **(May 2016)**

### Syllabus Topic : White Box Testing

Q. Write short note on white box testing. *(Refer section 6.4.1)* (5 Marks)

Q. What is static testing ? *(Refer section 6.4.1(A))* (2 Marks)

Q. State and explain different methods of static testing. *(Refer section 6.4.1(A))* (4 Marks)

Q. Compare walk through and formal inspection. *(Refer section 6.4.1(A))* (5 Marks)

Q. What are different approaches of structural testing ? *(Refer section 6.4.1(B))* (6 Marks)

Q. Explain code complexity testing and cyclomatic complexity of code with the help of suitable example. *(Refer section 6.4.1(B)(3))* (4 Marks)

Ex. 6.4.1 (10 Marks) **(May 2016)**

### Syllabus Topic : Black Box Testing

Q. What is black box testing ? *(Refer section 6.4.2)* (4 Marks)

Q. Explain the Equivalence class method of testing with one example. *(Refer section 6.4.2(A))* (6 Marks) **(May 2016)**

Q. What is equivalence partitioning ? *(Refer section 6.4.2(A))* (4 Marks)

Q. Explain graph based testing with suitable example. *(Refer section 6.4.2(B))* (8 Marks) **(Dec. 2016)**

Q. Differentiates between Black box testing and white box testing. *(Refer section 6.4.3)* (8 Marks) **(May 2017)**

Q. State the different phases of testing. (Refer section 6.5) (1 Mark) **(May 2016)**

### Syllabus Topic : Unit Testing

Q. Explain unit testing phase of testing. *(Refer section 6.5.1)* (2 Marks) **(May 2016)**

Q. List and explain the benefits of unit test. *(Refer section 6.5.1(A))* (6 Marks)

### Syllabus Topic : Integration Testing

Q. Explain integration testing phase of testing. *(Refer section 6.5.2)* (2 Marks) **(May 2016)**

Q. Explain integration testing. (Refer section 6.5.2) (6 Marks) **(Dec. 2016)**

Q. What are different types of integration testing? *(Refer section 6.5.2)* (2 Marks) **(May 2016)**

Q. Compare top-down and bottom up integration. *(Refer section 6.5.2(B))* (5 Marks)

Q. State the application of different integration methods. *(Refer section 6.5.2(E))*(4 Marks)

Q. Explain system testing. *(Refer section 6.5.3)* (4 Marks)

Q. What is need of system testing ? *(Refer section 6.5.3(A))* (4 Marks) **(May 2016)**

**Q.** Differentiate between functional and non-functional testing. *(Refer section 6.5.3(B))* (6 Marks)

**Q.** Briefly explain different types of functional system testing. *(Refer section 6.5.3(C))* (5 Marks)

### ☞ Syllabus Topic : GUI Testing

**Q.** Explain GUI testing. *(Refer section 6.5.4)* (4 Marks) **(May 2016)**

**Q.** What are different methods of GUI testing? *(Refer section 6.5.4(A))* (4 Marks)

**Q.** Explain the planning of GUI testing. *(Refer section 6.5.4(B))* (4 Marks) **(May 2016)**

**Q.** Explain the web based testing. *(Refer section 6.5.4(C))* (4 Marks) **(May 2016)**

### ☞ Syllabus Topic : User Acceptance Validation Testing

**Q.** What is acceptance testing ? *(Refer section 6.5.5)* (4 Marks)

**Q.** Briefly explain different criteria for acceptance testing. *(Refer section 6.5.1(A))* (5 Marks)

**Q.** Briefly explain different guidelines for acceptance testing. *(Refer section 6.5.5(B))* (5 Marks)

### ☞ Syllabus Topic : Performance Testing

**Q.** What is performance testing ? *(Refer section 6.7)* (2 Marks)

**Q.** Explain the need of performance testing. *(Refer section 6.7.1)* (4 Marks)

**Q.** Explain different activities of performance testing. *(Refer section 6.7.2)* (4 Marks)

**Q.** List few tools of performance testing. *(Refer section 6.7.3)* (2 Marks)

### ☞ Syllabus Topic : Test Cases and Test Plan

**Q.** Explain test cases and test plan ? *(Refer section 6.8)* (6 Marks)

### ☞ Syllabus Topic : Case studies expected for developing usability test plans and test cases

**Q.** With the help of an example, formulate test plan and test cases for real life application. *(Refer section 6.9)* (5 Marks)

❑❑❑