# Mod-4

## 4.1)

Modeling is an essential activity in software development, and it plays a crucial role in the design and documentation of software systems. The Unified Modeling Language (UML) is a widely adopted standard for modeling software systems, providing a comprehensive set of diagrams and notations to represent various aspects of the system.

Importance of Modeling:

1. **Visualization:** Models provide a visual representation of the system, making it easier to understand, communicate, and reason about complex concepts and relationships.
2. **Abstraction:** Modeling allows developers to focus on the essential aspects of the system by abstracting away unnecessary details, making it easier to manage complexity.
3. **Communication:** Models serve as a common language for stakeholders, facilitating effective communication and collaboration among team members, clients, and domain experts.
4. **Documentation:** Models document the design and architecture of the system, providing a reference for future maintenance and evolution.
5. **Analysis:** Models can be used to analyze and validate the system's behavior, identify potential issues, and explore alternative designs before implementation.
6. **Code Generation:** In some cases, models can be used as input for code generation tools, automating parts of the development process and ensuring consistency between the design and implementation.

## Design Concepts

### 6.4.1 Abstraction

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

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

### 6.4.2 Modularity

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

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

- Modularization should be such that the development can be planned easily, software increments can be defined and delivered, changes can be more easily accommodated and long term maintenance can be carried out effectively.
- **Meyer** defines five criteria that enable us to evaluate a design method with respect to its ability to define an effective modular system :

1. **Modular decomposability :** A design method provides a systematic mechanism for decomposing the problem into sub-problems. This reduces the complexity of the problem and the modularity can be achieved.
2. **Modular composability :** A design method enables existing design components to be assembled into a new system.
3. **Modular understandability :** A module can be understood as a standalone unit. It will be easier to build and easier to change.
4. **Modular continuity :** Small changes to the system requirements result in changes to individual modules, rather than system-wide changes.
5. **Modular protection :** An aberrant condition occurs within a module and its effects are constrained within the module.

### 6.4.4 Refinement

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

... are constrained within the module.

### 6.4.3 Architecture

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

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

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

### 6.4.6 Information Hiding

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

### 6.4.7 Functional Independence

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

## 4.4)

### 6.4.7.1 Cohesion

- With the help of cohesion the information hiding can be done.
- A cohesive module performs only "one task" in software procedure with little interaction with other modules. In other words cohesive module performs only one thing.
- Different types of cohesion are :

1. **Coincidentally cohesive** - The modules in which the set of tasks are related with each other loosely then such modules are called coincidentally cohesive.

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

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

### 6.4.7.2 Coupling

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

| Sr. No. | Coupling | Cohesion |
|---|---|---|
| 1. | Coupling represents how the **modules are connected** with other modules or with the outside world. | In cohesion, the cohesive module performs **only one thing.** |
| 2. | With coupling **interface complexity** is decided. | With cohesion, **data hiding** can be done. |
| 3. | The goal of coupling is to achieve **lowest coupling.** | The goal of cohesion is to achieve **high cohesion.** |
| 4. | Various types of couplings are - Data coupling, Control coupling, Common coupling and Content coupling. | Various types of cohesion are - Coincidental cohesion, Logical cohesion, Temporal cohesion, Procedural cohesion and Communicational cohesion. |

# Mod-5

## Reverse Engineering

Reverse Engineering: Reverse engineering is the process of analyzing and understanding an existing system or component to determine its design, architecture, and implementation details. The primary goal of reverse engineering is to extract knowledge or recreate design information from an existing system without having access to its original design documentation or source code.

The main tasks involved in reverse engineering include:

1. Disassembly: Breaking down the executable code or compiled components into a lower-level representation.
2. Code analysis: Analyzing the disassembled code to understand its structure, logic, and functionality.
3. Data structure recovery: Identifying and understanding the data structures and data formats used in the system.
4. Architecture recovery: Reconstructing the overall architecture and design of the system from the analyzed components.

**Ex:** • Security Auditing: Security researchers often reverse engineer software applications or systems to identify potential vulnerabilities, security flaws, or backdoors that could be exploited by malicious actors.

- Hardware Reverse Engineering: Reverse engineering techniques are used in the hardware domain to analyze and understand the design and functionality of electronic devices, integrated circuits, or firmware, particularly when the original design specifications are not available.

## Re-Engineering

Re-engineering: Re-engineering, on the other hand, is the process of modifying or restructuring an existing system to improve its quality, performance, or maintainability. It involves analyzing the current system, identifying areas for improvement, and then making the necessary changes.

The main tasks involved in re-engineering include:

1. Reverse engineering: Analyzing and understanding the existing system, as described above.

2. Restructuring: Modifying the system's architecture, design, or implementation to address identified issues or improve specific aspects.
3. Optimization: Enhancing the system's performance, scalability, or resource utilization.
4. Migration: Transitioning the system to a new platform, programming language, or technology stack.
5. Refactoring: Improving the internal structure and code quality of the system without changing its external behavior.

**Ex:**
- Cloud Migration: A company might re-engineer its on-premises software application to migrate it to a cloud environment, such as AWS or Azure. This could involve restructuring the application to take advantage of cloud-native services, scaling capabilities, and DevOps practices.

- Performance Optimization: A e-commerce website might undergo re-engineering to optimize its performance and handle increased traffic during peak seasons. This could involve caching strategies, database optimization, load balancing, or other architectural changes to improve responsiveness and throughput.

Software maintenance is the process of modifying, updating, or enhancing an existing software system after it has been deployed and is in operation. It is a crucial phase in the software development life cycle, as software systems often need to be maintained and evolved to meet changing requirements, fix bugs, improve performance, or adapt to new technologies and environments.

There are four main types of software maintenance:

1. Corrective Maintenance: Corrective maintenance involves fixing bugs, errors, or defects in the software that were not addressed during the initial development or testing phases. This type of maintenance is reactive and aims to restore the software to its intended functionality and behavior.
2. Adaptive Maintenance: Adaptive maintenance involves modifying the software to accommodate changes in the external environment or systems it interacts with. This could include adapting the software to work with new hardware, operating systems, libraries, or other software components. Adaptive maintenance ensures that the software remains compatible and functional in the face of environmental changes.
3. Perfective Maintenance: Perfective maintenance focuses on improving the existing software by enhancing its functionality, performance, usability, or maintainability. This type of maintenance is proactive and aims to improve the overall quality and capabilities of the software based on user feedback, new requirements, or technological advancements.
4. Preventive Maintenance: Preventive maintenance involves activities that aim to prevent future problems or minimize the likelihood of issues occurring in the software. This can include activities such as code refactoring, documentation updates, performance tuning, or implementing security patches. Preventive maintenance helps to improve the software's reliability, maintainability, and overall quality.

In addition to these four main types, there are other categories of software maintenance, such as:

- Emergency Maintenance: Addressing critical issues or failures that require immediate attention and resolution.